Resonant Thermoelectric Nanophotonics: Applications in Spectral and Thermal Sensing

Thesis by
Kelly Ann Weekley Mauser

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy in Applied Physics

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2019
Defended April 19, 2019

© 2019

Kelly Ann Weekley Mauser
ORCID: 0000-0001-9903-8559

All rights reserved

ACKNOWLEDGEMENTS

I would like to thank Prof. Harry Atwater for the scientific support and advice he has provided, and for welcoming me into his group. I would also like to thank Prof. Keith Schwab for allowing me to use his lab and for all the advice he has given me. Prof. Painter and Prof. Minnich (and their students) have also been helpful resources at various points in my time at Caltech.

Many people have helped me accomplish the work completed in this thesis. On the fabrication side, those who have given advice or who have been willing to discuss my many problems are Max, Dvin, Seyoon, Phil, Hal, Victor, Mike, Michelle, and Dagny, among others. Additionally, the KNI staff have been invaluable, and have made the KNI a great place to work, so thank you Guy, Alex, Matt, Nathan, Melissa, Bert, Barry, and Steven. I would especially like to thank Alex and Dvin (my sputter buddies) for being very accommodating. On the simulation side, Seyoon taught me a great deal, and Phil was excellent at keeping the computers up and running. I would especially like to thank Keith for teaching me everything about noise and noise thermometry and all kinds of different measurements, and for Constantine (Doug) Sideris for answering all of my circuit questions. Team safety has been a fun team to be a part of. Thanks Rebecca, Cris, Haley, Kelly 2.0, and Eowyn. I wish you the best in continuing to fight the unwinnable war.

I would like to thank Seyoon Kim (my very best friend), for being a constant source of support, and for a lot of fun times. Not only is he a great friend, but he is a great scientist and has very helpfully and directly told me I was wrong a lot (he was usually right).

I would like to thank Jacob, Ellen, Max, Charlie, and Cedric, who helped me survive Caltech classes and are great friends and programming resources. Keeping me sane outside of lab, I would like to thank Anya, Yury, Ruzan, Nick, Dylan, Andrew, Nate, Fred, Soichi, Ryan (I guess), Roberta, Emily, and Cora. I would also like to thank the amazing administrators in our department, including Jonathan, Angie, Christy, Jennifer, Tiffany, Connie, Lyann, and Liz. They have been so helpful over the years and always so kind.

I would also like to thank Nathan for being a great partner and travel buddy during grad school. I couldn't have done this without his support.

Finally, and most importantly, I would like to thank Mom, Dad, and Paul for always

supporting me through the good and the bad.

ABSTRACT

Plasmon excitation enables extreme light confinement at the nanoscale, localizing energy in subwavelength volumes and thus can enable increased absorption in photovoltaic or photoconductive detectors. Nonetheless, plasmon decay also results in energy transfer to the lattice as heat which is detrimental to photovoltaic detector performance. However, heat generation in resonant subwavelength nanostructures also represents an energy source for voltage generation, as we demonstrate in the first part of this thesis via design of resonant thermoelectric (TE) plasmonic absorbers for optical detection. Though TEs have been used to observe resonantly coupled surface plasmon polaritons in noble-metal thin films and microelectrodes, they have not been employed previously as resonant absorbers in functional TE nanophotonic structures.

We demonstrate nanostructures composed of TE thermocouple junctions using established TE materials – chromel/alumel and bismuth telluride/antimony telluride – but patterned so as to support guided mode resonances with sharp absorption profiles, and which thus generate large thermal gradients upon optical excitation and localized heat generation in the TE material. Unlike previous TE absorbers, our structures feature tunable narrowband absorption and measured single junction responsivities 4 times higher than the most similar (albeit broadband) graphene structures, with potential for much higher responsivities in thermopile architectures. For bismuth telluride – antimony telluride single thermocouple structures, we measure a maximum responsivity of 38 V/W, referenced to incident illumination power. We also find that the small heat capacity of optically resonant TE nanowires enables a fast, 3 kHz temporal response, 10-100 times faster than conventional TE detectors. We show that TE nanophotonic structures are tunable from the visible to the MIR, with small structure sizes of 50 microns x 100 micons. Our nanophotonic TE structures are suspended on thin membranes to reduce substrate heat losses and improve thermal isolation between TE structures arranged in arrays suitable for imaging or spectroscopy. Whereas photoconductive and photovoltaic detectors are typically insensitive to sub-bandgap radiation, nanophotonic TEs can be designed to be sensitive to any specific wavelength dictated by nanoscale geometry, without bandgap wavelength cutoff limitations. From the point of view of imaging and spectroscopy, they enable integration of filter and photodetector functions into a single structure. Other thermoelectric nanophotonic motifs are also explored.

Generating localized, high electric field intensity in nanophotonic and plasmonic devices has many applications, from enhancing chemical reaction rates, to thermal radiation steering, to chemical sensing, and to photovoltaics. Along with a strongly localized electric field comes a temperature rise in non-lossless photonic materials, which can affect reaction rate, photovoltaic efficiency, or other properties of the system. Measuring temperature rises in nanophotonic structures is difficult, and methods commonly employed suffer from various limitations, such as low spatial resolution (Fourier transform infrared microscopy), bulky and expensive setups (scanning thermal microscopy), intrusive methods that interfere with nanophotonic structures (Pt resistive thermometry), or the need for specialized materials (temperature dependent photoluminescence).

In the second part of this thesis, we overcome these limitations with the first-ever demonstration of temperature measurements of nanophotonic structures by employing both room temperature noise thermometry and the thermoelectric effect under ambient conditions without external probes by utilizing the properties of the materials that make up the nanophotonic structure itself. We have previously estimated the Δ T in a nanophotonic device using the thermoelectric effect, but could not determine the absolute temperature of the system. In the application we will discuss, the absolute electron temperature of the nanophotonic material itself is measured. Because Johnson-Nyquist noise is material independent and is a fundamental measure of absolute temperature, there is theoretically no need for calibration as in the case of resistive thermometry. To measure the temperature rise of a nanophotonic resonant region remotely, the Seebeck coefficient of the material is first carefully measured using noise thermometry, then the thermoelectric voltage generated in the nanophotonic materials themselves is measured from electrical leads spanning the resonantly excited region. To accomplish this, we have developed a metrology technique capable of simultaneously measuring electrical noise at two locations on the nanophotonic structure as well as the electrical potential between the two points, under chopped laser illumination that heats the structure via nanophotonic absorption, thus providing drift-corrected light on/off temperature information.

PUBLISHED CONTENT AND CONTRIBUTIONS

Mauser, Kelly W et al. (2017). "Resonant thermoelectric nanophotonics". In: *Nature nanotechnology* 12.8, p. 770. DOI: https://doi.org/10.1038/nnano.2017.87.

K.A.W.M. participated in the conception of the project, fabricated the structures, performed the measurements and simulations, analyzed the data, and participated in the writing of the manuscript.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

Loss of incident photons to heat is often undesirable in light-detection or energy conversion technologies. For instance, in photovoltaic systems, thermalization of above-bandgap electrons to the band edge is lost power, and thermalization of valence band electrons into the conduction band leads to decreased sensitivity via excess noise in photodiode-based detectors. This energy lost to heat, however, can be harnessed as a detection mechanism. Using thermoelectric materials, which convert heat into electrical power, with the help of nanophotonics to capture more light in a smaller volume and thereby decrease response times, and by measuring the energy of the electrons directly via noise thermometry, a number of new detectors and applications can be imagined.

The following chapter is an introduction to all of the necessary concepts in this thesis, including nanophotonics, thermoelectricity, and noise thermometry. Applications combining these various topics will appear later in the thesis.

1.1 Nanoscale light-matter interactions

Light interacting with sub-wavelength geometries behaves differently than with macroscale objects. While the same Maxwell's equations apply in both cases, simpler approximations can be used to solve macroscale optics problems, such as ray tracing. On the other hand, nanoscale structures with dimensions on the order of the wavelength of incident light, cannot use such simple laws, and Maxwell's equations must be directly solved to determine the behavior of light within each structure. Maxwell's equations are written as follows:

$$\nabla \times \mathbf{E}(\mathbf{r}, t) = -\frac{\partial \mathbf{B}(\mathbf{r}, t)}{\partial t},$$

$$\nabla \times \mathbf{H}(\mathbf{r}, t) = \frac{\partial \mathbf{D}(\mathbf{r}, t)}{\partial t} + \mathbf{j}(\mathbf{r}, t),$$

$$\nabla \cdot \mathbf{D}(\mathbf{r}, t) = \rho(\mathbf{r}, t),$$

$$\nabla \cdot \mathbf{B}(\mathbf{r}, t) = 0,$$

where **E** is electric field, **D** is electric displacement, **H** is magnetic field, **B** is magnetic inductance, **j** is current density, and ρ is charge density. The relationship between electric field and electric displacement, and between the magnetic field and magnetic induction is

$$\mathbf{D}(\mathbf{r}, t) = \varepsilon_0 \mathbf{E}(\mathbf{r}, t) + \mathbf{P}(\mathbf{r}, t),$$

$$\mathbf{H}(\mathbf{r}, t) = \frac{1}{\mu_0} \mathbf{B}(\mathbf{r}, t) - \mathbf{M}(\mathbf{r}, t),$$

where **P** is the polarization, **M** is the magnetization, ε_0 is the permittivity of free space, and μ_0 is the permeability of free space.

These equations are often intractable analytically for a given nanoscale structure and must be solved numerically. Occasionally, simplified models can be used to approximate nanophotonic behavior, such as coupled mode theory (Haus and W. Huang, 1991).

Interactions of light with subwavelength structures

Interesting effects can arise when light interacts with subwavelength particles. For example, a macroscale piece of gold appears yellow under a white light source, but if this piece of gold were separated into 30 nm spheres and suspended in a transparent liquid, the gold would appear red (Njoki et al., 2007). Prior to modern computers, calculations done with light used some variation of ray-tracing to determine light trajectories for structures much larger than the wavelength of light.

Only the very simplest "nanophotonic" structures could be designed without a computer. One example is the Salisbury screen (Salisbury, 1952), first designed to absorb radar. It consists of a dielectric of thickness equal to 1/4 of the wavelength of interest (when traveling in the dielectric), sandwiched between a backreflector and a thin absorbing layer. This design can absorb close to 100% of light of a specific wavelength. With today's computing power, Maxwell's equations can be solved numerically, opening up a wide range of previously unsolvable (or too time-consuming to solve) sub-wavelength geometries for study.

The material composing the subwavelength structure as well as the incident wavelength of light can have an enormous impact on the outcome of the light-matter interaction. For instance, light with energy close to a resonant lattice vibration

energy (phonon) or a resonant electronic oscillation energy (plasmon) will interact with the material differently than a photon with energy much greater or less than these features. Materials have intrinsic (bulk) electron oscillation energies, which can be shifted or modified by geometry; the edges of a nanostructure can introduce boundary conditions which support different oscillating modes of electrons. Metals, due to the large populations of freely moving electrons in their conduction band, are especially affected by structure size in this way. Semiconductors have far fewer electrons in their conduction bands and have additional interactions with light in interband transitions between the valence and conduction band. Semiconductor-light interactions close to or above the band edge are often more complicated than metal-light interactions to intuit or model in simple ways. Insulators (which can be thought of as semiconductors with very large bandgaps), can have phononic interactions with lower energy light (similar to the semiconductors and metals) or electronic interactions with very high energy light. In essence, different materials in the same nanophotonic geometry can behave in radically different ways when exposed to light.

Metals can be used in plasmonic applications to focus light in very small mode volumes, but suffer from high loss (which can be beneficial in some applications (X. Huang et al., 2008)). Insulators are less adept at concentrating light into small volumes, but have very low loss and have found uses in metalens applications (Khorasaninejad et al., 2016). Semiconductors (and semimetals) find uses related to their bandgap. Either used as an insulator below their band edge (i.e. in silicon photonics for low-loss waveguides), or their bandgap is used to segregate and transport electrons (photovoltaics, thermoelectrics).

Not only does the atomic composition of the material play a role, but the crystal structure is also important. For example, VO_2 has an insulating phase below 68°C and metallic phase above, yielding correspondingly different optical properties (which has recently found uses in optical switching applications, e.g. (Briggs, Pryce, and Harry A. Atwater, 2010)). Silicon photovoltaics perform remarkably better when in very crystalline form, due to higher carrier lifetimes (i.e. electrons scatter and recombine at grain boundaries) (Imaizumi et al., 1997).

1.2 Thermoelectrics

The thermoelectric effect was first discovered by Thomas Seebeck around 1821 (Zhang and Zhao, 2015). The uses of thermoelectrics have been limited, how-

ever, due to the difficulty of increasing the efficiency above 7% (Vining, 2009). Thermoelectrics have found uses in niche applications, however, including many space-based applications. A radioactive thermoelectric generator is being used to power the Mars Curiosity rover as well as other deep space probes which cannot rely solely on solar power due to either dust storms or their distance from the sun. Additionally, thermoelectrics are much less sensitive to radiation damage (H. Wang and Leonard, 2017), whereas silicon solar cells have efficiency degradation when exposed to radiation (Treble, 1962) . In fact, thermoelectric performance can actually be enhanced by crystal defects, due to phonon scattering at crystal boundaries and corresponding lowering of thermal conductivity (Koumoto and Mori, 2013). As additional evidence of the robustness of thermoelectrics against radiation, thermoelectric materials are in practice attached adjacent to a decaying radioactive core in radioactive thermoelectric generators.

The basic operating principle behind thermoelectric emf generation is if one end of a material is heated, the majority carriers will "diffuse" toward the cold end of the material (the minority carriers will also "diffuse" toward the cold end, but the majority carrier "diffusion" produces a larger effect), generating a voltage difference that is proportional to the temperature difference by the Seebeck coefficient, α. α is material dependent and temperature dependent (Zhang and Zhao, 2015). See Figure 1.1 for a heavily simplified explanation of thermoelectric operation, including the operature of a thermocouple.

If instead of heating one end, a current is passed through the thermoelectric element, the temperature will fall at one end of the device and rise at the other end with polarity depending on the sign of the Seebeck coefficient. This is called the Peltier effect. The next section will discuss the thermoelectric effect theory in more detail.

Thermoelectric effect theory

The thermoelectric effect can be derived from the electronic Boltzmann transport equation, and is the analog of thermal conductivity in the phononic Boltzmann transport equation. In reality, calculations rarely predict the Seebeck coefficient of materials accurately. This is in part due to imperfect crystal structure in experiment and the effect of grain boundaries, among other factors. The Seebeck coefficient can be highly dependent on deposition method and stoichiometry, (see Table 3.2). A brief theory of the thermoelectric effect follows, from (W. Wang and Z. M. Wang, 2014).

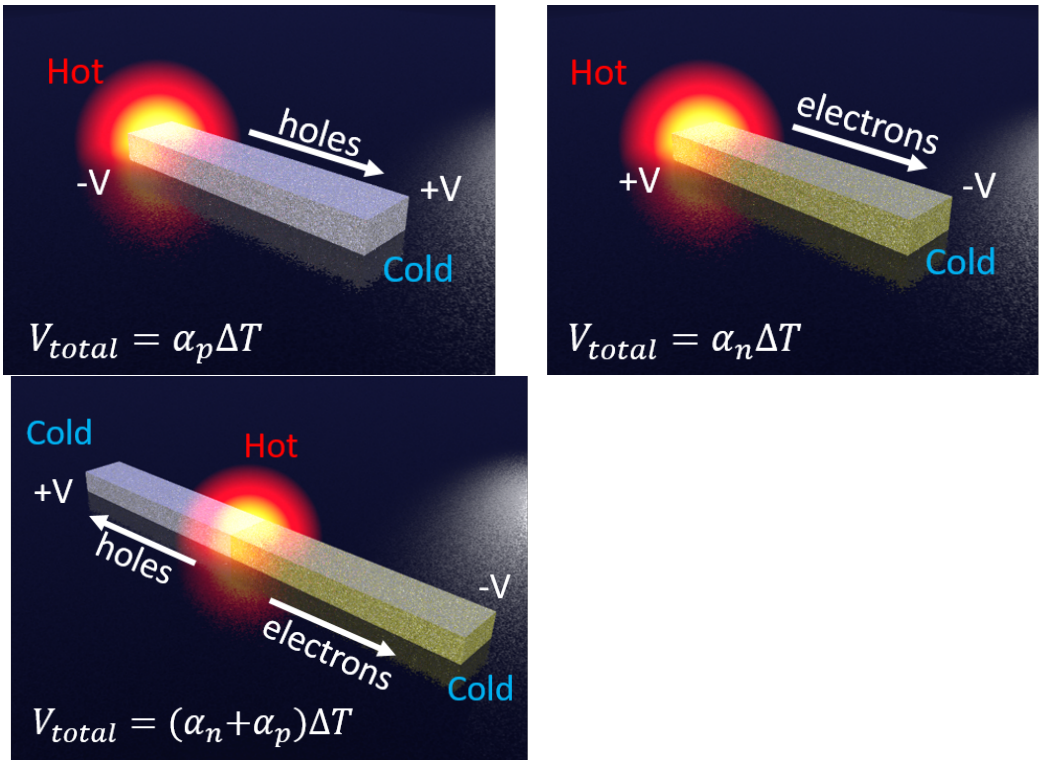

Figure 1.1: Simplified version of thermoelectric operation for p-type (top left), n-type (top right), and a thermocouple (bottom).

The electrons in a material obey Fermi-Dirac statistics with a distribution function, f_0 of

$$f_0 = \frac{1}{e^{(\epsilon(\mathbf{k})-\mu)/k_b T} + 1}, \tag{1.1}$$

where $\epsilon(\mathbf{k})$ is the electron energy, μ is the chemical potential, or the Fermi potential at 0 K, k_B is the Boltzmann constant, and T is the absolute temperature in Kelvin. In a material, this distribution function becomes dependant on position, $\mathbf{r}$, wave number (which depends on the band structure), $\mathbf{k}$, and time, t. In equilibrium, this function is

$$f(\mathbf{r},\mathbf{k}) = \frac{1}{e^{(\epsilon(\mathbf{k})-\mu(\mathbf{r}))/k_b T(\mathbf{r})} + 1}. \tag{1.2}$$

We can find the time dependence of this distribution function by taking the time derivative and using the chain rule,

$$\frac{df}{dt} = \nabla_{\mathbf{r}} f \cdot \frac{d\mathbf{r}}{dt} + \frac{1}{\hbar}\nabla_{\mathbf{k}} f + \frac{d\mathbf{p}}{dt} + \frac{\partial f}{\partial t} = \mathbf{v} \cdot \nabla_{\mathbf{r}} f + \frac{1}{\hbar}\nabla_{\mathbf{k}} f \cdot \mathbf{F}_a + \frac{\partial f}{\partial t} = 0, \quad (1.3)$$

where this equation equals zero because the total number of states is constant in time. In the above equation, F_a is the set of applied external and internal (e.g. impurities, phonons, etc...). In the right-most form of the equation above, the last term is due to internal fields and is the scattering term, and the first two terms are due to external forces, or are drift terms. This gives, in the time independant case where $\partial f/\partial t = 0$,

$$\frac{\partial f}{\partial t} = \left(\frac{\partial f}{\partial t}\right)_{drift} + \left(\frac{\partial f}{\partial t}\right)_{scattering} = 0. \quad (1.4)$$

While the drift term is classical and deterministic, the scattering term is quantum mechanical and non-deterministic, and is the difficult term to deal with as it requires scattering theory to solve. One method of dealing with this scattering term is by using the relaxation time approximation, where it is assumed that the change in the distribution function with time is proportional to the difference between the current distribution state and f_0, the equilibrium distribution state, with proportionality constant $1/\tau(\mathbf{k})$, where the relaxation time τ is how fast the system returns to equilibrium. This approximation assumes the system is not far from the equilibrium state. This approximation gives

$$\left(\frac{\partial f}{\partial t}\right)_{scattering} = -\frac{f - f_0}{\tau(\mathbf{k})}. \quad (1.5)$$

Using the time-independant form of the Boltzmann Equation where the relaxation time approximation is used and assuming spatial homogeneity and $\mathbf{F}_a = e\mathbf{E}$, the Boltzmann equation can be analytically solved as

$$f(\mathbf{k}) \approx f_0(\mathbf{k}) - \frac{e}{\hbar}\tau(\mathbf{k})\mathbf{E} \cdot \nabla_{\mathbf{k}} f. \quad (1.6)$$

If we are interested in linear $\mathbf{E}$-field phenomena, then we can replace f on the right hand side with f_0, then taylor expand about $\mathbf{k}$, giving

$$f(\mathbf{k}) \approx f_0(\mathbf{k}) - \frac{e}{\hbar}\tau(\mathbf{k})\mathbf{E}\cdot\nabla_{\mathbf{k}} f_0 \approx f_0\left(\mathbf{k} - \frac{e}{\hbar}\tau(\mathbf{k})\mathbf{E}\right). \tag{1.7}$$

In a similar way, taking away the spatial homogeneity condition, the Boltzmann equation can be linearized as

$$\mathbf{v}\cdot\nabla_r f_0 + \frac{e}{\hbar}\mathbf{E}\cdot_k f_0 = -\frac{f(\mathbf{r},\mathbf{k}) - f_0(\mathbf{r},\mathbf{k})}{\tau(\mathbf{k})} \equiv -\frac{f^{(1)}(\mathbf{r},\mathbf{k})}{\tau(\mathbf{k})}, \tag{1.8}$$

where $f^{(1)}$ is a first order term in a Taylor expansion. Because we know the form of f_0, the gradients in the above expression can be simplified and rewritten (see (W. Wang and Z. M. Wang, 2014)), so we find

$$f^{(1)}(\mathbf{r},\mathbf{k}) = -\frac{\partial f_0}{\partial \epsilon}\tau(\mathbf{k})\mathbf{v}\cdot[\nabla(e\phi + \mu) + (\epsilon - \mu)\nabla \ln T], \tag{1.9}$$

where $\mathbf{E} = -\nabla\phi$, where ϕ is the electrostatic potential. Next, we look a the current density, $\mathbf{j}$,

$$\mathbf{j} = \frac{e}{4pi^3}\int \mathbf{v} f(\mathbf{r},\mathbf{k}) d^3\mathbf{k}. \tag{1.10}$$

Substituting in $f^{(1)}$, and assuming an isotropic medium (no $\mathbf{r}$ dependence), and introducing the density of states $D(\epsilon)$,

$$\mathbf{j} = e\mathcal{L}_{11}\left(e\mathbf{E} - T\nabla\frac{\mu}{T}\right) - e\mathcal{L}_{21}\nabla \ln T, \tag{1.11}$$

where

$$\mathcal{L}_{ij} = \frac{4}{3m^*}\int\left(-\frac{\partial f_0}{\partial \epsilon}\right)\tau^j \epsilon^i D(\epsilon) d \tag{1.12}$$

and the energy flux is given by

$$\mathbf{j}_Q = \frac{e}{4pi^3}\int \epsilon(\mathbf{k})\mathbf{v}f(\mathbf{r},\mathbf{k})d^3\mathbf{k} = e\mathcal{L}_2 1\left(e\mathbf{E} - T\nabla\frac{\mu}{T}\right) - \mathcal{L}_{31}\nabla \ln T. \tag{1.13}$$

From $\mathcal{L}$, we can extract the electrical conctivity, $\sigma = e^2\mathcal{L}_{11}$. If $\mathbf{j} = 0$ (no currents) and there is no concentration gradient ($\nabla\mu = 0$), then from Equation 1.11,

$$\mathbf{E} = \frac{\mathcal{L}_{21} - \mu\mathcal{L}_{11}}{e\mathcal{L}_{11}T}\nabla T, \tag{1.14}$$

i.e. the Seebeck coefficient, α, is

$$\alpha = \frac{\mathcal{L}_{21} - \mu\mathcal{L}_{11}}{e\mathcal{L}_{11}T}. \tag{1.15}$$

If we put the electrical current $\mathbf{j}$ in Equation 1.13, we find

$$\mathbf{j}_Q = \Pi\mathbf{j} - \kappa_e\nabla T, \tag{1.16}$$

where the Peltier coefficient is (again with no gradient in chemical potential),

$$\Pi = \frac{\mathcal{L}_{21}}{e\mathcal{L}_{11}} = \alpha T, \tag{1.17}$$

and the electron thermal conductivity is

$$\kappa_e = \frac{\mathcal{L}_{21}^2 - e\mathcal{L}_{31}\mathcal{L}_{11}}{e\mathcal{L}_{11}}. \tag{1.18}$$

For a metal or degenerate semiconductor, we can find using additional terms in the Fermi integral, that

$$\alpha = \frac{\pi^2 k_B^2 T}{3e\epsilon_F} \propto \frac{\partial D(\epsilon)}{\partial\epsilon}\bigg|_{\epsilon=\epsilon_F}. \tag{1.19}$$

The Seebeck coefficient for metals is typically a few μV/K. For a semiconductor with assumed $\tau(\epsilon) = \tau_0 \epsilon^\lambda$,

$$\alpha = k_B e \left[\frac{5}{2} + \lambda - \ln \frac{N_c}{n} \right], \tag{1.20}$$

where n, N_c are carrier concentration terms. This equation gives general guidelines for semiconductor Seebeck coefficients, but can be drastically different from experimental data, especially with multiple bandgaps near the Fermi level (W. Wang and Z. M. Wang, 2014; Koumoto and Mori, 2013). Seebeck coefficients of semiconductors can be as high as 100's of μV/K.

The thermoelectric effect can also be derived with thermodynamic arguments with the first and second Kelvin relations (see http://www.eng.fsu.edu/ dommelen/quantum/style_a/nt_pelt.html for example).

In the limiting case where no current is flowing, only the Seebeck coefficient is needed to determine voltage generation. When the system is isothermal (with good heat sinking, for instance) and only current is flowing, the Peltier coefficient is the relevant quantity. When current is flowing and the system is not isothermal, both $\mathbf{j}$ and $\mathbf{j}_Q$ are coupled and need to be solved together.

The Seebeck coefficient is just one contribution to thermoelectric efficiency. The figure of merit for thermoelectric efficiency is

$$zT = \frac{\alpha^2 \sigma T}{\kappa}, \tag{1.21}$$

where "good" thermoelectric materials have a zT of around 1 or higher, corresponding to about 4-5% efficiency or higher (Zhang and Zhao, 2015).

1.3 Noise thermometry

The presence of the random motion of electrons in circuits was first suggested by Albert Einstein in his 1906 paper on Brownian Motion (Einstein, 1905). Johnson was the first to measure the presence of this noise (John B Johnson, 1927; John Bertrand Johnson, 1928), and a year later Nyquist developed the theory to explain Johnson's findings (Nyquist, 1928). Nyquist's theorem was later generalized into the fluctuation dissipation theorem (Rep Kubo, 1966), which governs a wide range of processes.

The first noise thermometer, unknowingly, was created by Dicke (Dicke, 1946) in 1946 to measure thermal radiation at microwave frequencies. A general overview of noise can be found in (White et al., 1996). More recently, noise thermometry was used in observing the quanta of thermal conductance (Keith Schwab et al., 2000), and is being used by NIST to redefine the Boltzmann constant (Qu et al., 2013).

Noise is often measured in noise spectral density (NSD), in units of $V/\sqrt{Hz}$ or $A/\sqrt{Hz}$. Another important measure is the power spectral density (PSD), which is the square of the NSD. There are a number of different noise sources, including Johnson noise from resistive elements, in which $NSD = \sqrt{4Rk_BT}$, where R is resistance, k_B is the Boltzmann constant, and T is temperature in Kelvin. Shot noise is given by $NSD = 2qI$, where q is the electron charge, and I is the current. Both Johnson and shot noise appear as a flat background at all frequencies. 1/f noise is another source of noise in semiconductor devices, which occurs at lower frequencies with a $1/f^{\beta}$ shape, where β is component dependent. Other noise sources, such as avalanche noise, exist, but in noise thermometry, Johnson noise is the noise source of interest and what we will focus on in the next section.

Theory of Johnson-Nyquist noise

The derivation in this section is summarized from (Buckingham, 1983), and is an alternative derivation of Johnson noise than that foundin Nyquist's original paper.

Noise in electronic device is stochastic, and assumed to be small fluctuations about the mean, so "small signal" theory can be used. In other words, if an electron randomly moves away from the equilibrium state, on average, its next move will be back toward equilibrium. Noise is also statistically stationary, meaning that the statistical properties are independent of epoch where measured. Shot noise and Johnson noise (which will be derived), follow a normal distribution about the mean. These noise sources can also be approximated well by a chain of uncorrelated delta functions, called an impulse process.

The Fourier transform is used in measuring the power spectral density of noise. In order for a function to be Fourier transformable, its absolute value integrated over all of time must be finite. For a noise signal, this is not the case, as the noise lasts for all time. But while the total energy is infinite, the power, or energy per time is finite, so Fourier analysis can be conducted as long as a section of the signal is analyzed, i.e. the signal, $f(t)$, is 0 at all times except for $-\frac{\mathcal{T}}{2} < t < \frac{\mathcal{T}}{2}$, where $\mathcal{T}$ is finite. If we look at the inverse Fourier transform of $f(t)$,

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega, \tag{1.22}$$

we see that the kernel, $e^{i\omega t}$ is a harmonic wave, so $F(i\omega)$ must be an amplitude spectrum. By Plancherel's theorem (assuming $f(t)$ is real over the interval $[-\frac{\mathcal{T}}{2}, \frac{\mathcal{T}}{2}]$ and zero everywhere else),

$$\int_{-\infty}^{\infty} [f(t)]^2 dt = \frac{1}{2\pi} \int_{0}^{\infty} |F(\omega)|^2 d\omega. \tag{1.23}$$

Both sides of this equation equal the total energy in $f(t)$, and therefore $|F(\omega)|^2$ is the energy density (with units energy/Hz), and dividing by $\mathcal{T}$ will give the average power density (with units power/Hz). Thus, the average power spectral density is given by

$$\overline{S_x(\omega)} = \lim_{\mathcal{T}\to\infty} \frac{2\overline{|F(\omega)|^2}}{\mathcal{T}}. \tag{1.24}$$

As an aside, which can be read about further in (Buckingham, 1983), the PSD and autocorrelation function of a process are related by the Wiener-Khintchine theorem. The electronic and thermal autocorrelation functions together with their cross-correlation function can be used to obtain the kinetic coefficients of electronic thermal conductivity, electrical resistivity, and the Seebeck coefficient through Green-Kubo theory (Ryogo Kubo, Yokota, and Nakajima, 1957; Garrity, 2011). This appears to be an experimentally understudied method of measuring Seebeck coefficient via equilibrium measurements, which should be further explored.

Continuing with the summarized derivation from (Buckingham, 1983), thermal and shot noise can be modeled as random pulse trains,

$$f(t) = \sum_{n=1}^{N} a_n g(t - t_n), \tag{1.25}$$

where $g(t)$ is the pulse shape function, a_n is the amplitude of the nth pulse, t_n is the time of the nth pulse, and N is the number of pulses in the time interval $\mathcal{T}$. t_n is Poisson distributed since the events are independent, so the expected value is

$$\overline{f(t)} = v\overline{a}\int_{-\infty}^{\infty} g(t)dt, \tag{1.26}$$

where v is the mean number of pulses per second and $\overline{a}$ is the mean amplitude of the pulses. The Fourier transform of $f(t)$ is

$$F(\omega) = G(\omega)\sum_{n=1}^{N} a_n e^{-i\omega t_n}. \tag{1.27}$$

From Equation 1.24, we can find the PSD of $f(t)$ as

$$\overline{S_x(\omega)} = \lim_{\mathcal{T}\to\infty} \frac{2\overline{|G(\omega)|^2}}{\mathcal{T}} \sum_{n=1}^{N}\sum_{m=1}^{N} \overline{a_n a_m e^{-i\omega(t_n - t_m)}}. \tag{1.28}$$

Only the $n = m$ terms survive assuming the a_n are distributed symmetrically about zero, yielding (with $v = N/\mathcal{T}$),

$$\overline{S_x(\omega)} = 2v\overline{a^2}|G(\omega)|^2, \tag{1.29}$$

which is Carson's theorem. We now apply this to a resistor with cross-sectional area A and length L. An electron travel length between collisions in the resistor is l_f. It is assumed that when an electron is "pulsed" out of equilibrium, it will relax to equilibrium. When the pulse is applied, it can be imaged that an instantaneous capacitor made of two charged plates with charge density $\pm q/A$ separate by length l_f form, then come back together. The equivalent circuit is shown in Figure 1.2,

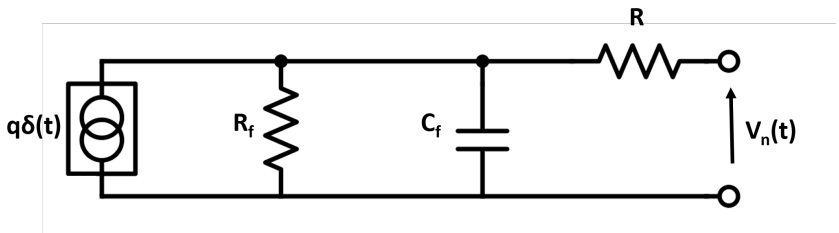

Figure 1.2: Equivalent circuit of a single "pulse" of an electron in a resistor.

where R_f and C_f are the resistance and capacitance between the two sheets, R is the resistance of the total resistor, and $q\delta(t)$ is the "pulse". The voltage, $v_n(t)$ seen at the nodes can be found with the RC circuit equation

$$C\frac{dv_n(t)}{dt} = -\frac{v_n(t)}{R} + q\delta(t), \tag{1.30}$$

where $C = \frac{C_f l_v}{L}$. Fourier transforming both sides yields

$$V_n(\omega) = \frac{q l_f (R/L)}{(1 + i\omega\tau)}, \tag{1.31}$$

where $\tau = RC$, which is very small in this case. From Carson's theorem (Equation 1.29), then

$$\overline{S_v(\omega)} = \frac{2\nu q^2 (R/L)^2 \overline{l_f^2}}{(1 + \omega^2\tau^2)} = \frac{4k_B T R}{(1 + \omega^2\tau^2)}, \tag{1.32}$$

where k_B is the Boltzmann constant and T is temperature, and the right hand term comes after making substitutions for ν, R, and μ described in (Buckingham, 1983), where this entire derivation is summarized from. In resistors, τ is negligible, so we find the familiar equation for PSD,

$$\overline{S_v(\omega)} = PSD = 4Rk_B T. \tag{1.33}$$

The noise spectral density (NSD) with units $V/\sqrt{\text{Hz}}$ is then

$$\sqrt{\overline{S_v(\omega)}} = NSD = \sqrt{4Rk_B T}. \tag{1.34}$$

This description is valid for metallic-like materials for $kT >> \hbar\omega$, where $\hbar$ is Planck's constant divided by 2π and ω is the signal frequency. Nyquist (Nyquist, 1928) noted that when $\hbar\omega$ becomes comparable or larger than kT, kT should be replaced and the zero-point energy term should be added, yielding

$$\overline{S_v(\omega)} = 4R\left(\frac{1}{2}\hbar\omega + \frac{\hbar\omega}{e^{\frac{\hbar\omega}{kT}} - 1}\right). \tag{1.35}$$

In this thesis, we are in the classical regime and will not need to use Equation 1.35.

Precision of noise thermometry

The temperature resolution we can achieve is dictated by the Dicke radiometer relation(Dicke, 1946), which is given by

$$\Delta T = \frac{T + T_{cirq}}{\sqrt{\Delta f t}}, \tag{1.36}$$

where ΔT is the temperature resolution, T is the temperature of the resistor in Kelvin, T_{cirq} is the temperature of the circuit, Δf is the measurement bandwidth in Hz, and t is the measurement time, in seconds.

1.4 Scope of thesis

This thesis explores performance and properties of nanophotonic structures composed of thermoelectric materials, both theoretically and experimentally. The chapters are organized as follows:

- *Chapter II – Design considerations and simulations of nanophotonic thermoelectric devices*

  This section will give an overview of what should be considered when designing thermoelectric nanophotonic structures in order to maximizing performance. We will discuss how the simulations must be developed, as well as the impact of materials choices on performance, and the interplay between responsivity and thermal time constant. Simulations of various nanophotonic thermoelectric devices for sensing applications will be presented.

- *Chapter III – Hyperspectral detector application*

  One of the devices described in Chapter II will be explored experimentally in this section for its potential as a hyperspectral detector.

- *Chapter IV – Nanoscale temperature measurements with noise thermometry*

  In this chapter, the theoretical considerations will be introduced for performing noise measurements on thermoelectric nanophotonic devices, including circuit design and component selection.

- *Chapter V – Noise thermometry results*

  The data analysis technique, as well as noise thermometer circuit tests and preliminary results will be introduced in this section.

- *Chapter VI – Conclusion*

  This thesis will be summarized and an outlook will be given for thermoelectric nanophotonic device performance for both spectral and thermometry applications.

- *Appendices*

  The appendices provide code for data analysis using the FFTW package in C, as well as description of the Python generated GUIs for instrument control in data collection.

Chapter 2

DESIGN CONSIDERATIONS AND SIMULATIONS OF NANOPHOTONIC DEVICES

2.1 Design considerations in nanophotonic thermoelectric devices

Plasmon excitation enables extreme light confinement at the nanoscale, localizing energy in subwavelength volumes, and thus can enable increased absorption in photovoltaic or photoconductive detectors (Harry A Atwater and Polman, 2010). Nonetheless, plasmon decay also results in energy transfer to the lattice as heat which is detrimental to photovoltaic detector performance (Skoplaki and Palyvos, 2009). However, heat generation in resonant subwavelength nanostructures also represents an energy source for voltage generation, as we demonstrate here via design of resonant thermoelectric plasmonic absorbers for optical detection. Though thermoelectrics have been used to observe resonantly coupled surface plasmon polaritons in noble-metal thin films and microelectrodes (Innes and Sambles, 1985; Weeber et al., 2011) and have been explored theoretically for generation of ultrafast intense magnetic pulses in a dual-metal split ring resonator (Tsiatmas et al., 2013), they have not been employed as resonant absorbers in functional thermoelectric nanophotonic structures. Previously, non-narrowband photodetection has been demonstrated through the photothermoelectric effect in gated graphene structures(Cai et al., 2014; Xu et al., 2009) and the laser heating of nanoantennas and micropatterned materials (Russer et al., 2015; Szakmany et al., 2015; A. S. Gawarikar, R. P. Shea, and J. J. Talghader, 2013b; F Völklein and A Wiegand, 1990; Hsu et al., 2015), all shown to be promising for infrared to THz broadband detection. Typical responsivities of the graphene structures are around 10 V/W for infrared and THz detectors, relative to incident (not absorbed) power, with a time response ranging from 23 ms to nearly 10 ps. Responsivities of non-graphene detectors range from 10's of V/W to nearly 7,000 V/W (A. S. Gawarikar, R. P. Shea, and J. J. Talghader, 2013b) for thermopiles made of many thermocouples of up to mm sizes. The response time of these structures range from 10's to 100's of ms, though ns response times have been predicted8 for nanoantenna structures. High-figure-of-merit thermoelectrics have been investigated as solar power generators, but the light absorption process was entirely separate from the thermoelectric functionality and relied on black carbon absorbers (Kraemer et al., 2011) or solar concentrators (Amatya and Ram, 2010).

2.2 Scale of nanophotonic vs. thermal simulations

In our applications, nanophotonic electromagnetic fields extend over the sub-nanometer to hundreds of nanometers scale, whereas temperature gradients occur on the 10's of microns scale. In simulations, this creates difficulties with scales and mesh sizes, as the mesh sizes needed for accurate nanophotonic simulations are impractical to use in thermal simulations due to memory constraints. So while coupled physics could be used in COMSOL to solve for both the thermal properties and optical properties simultaneously, this was computationally prohibitive in our applications, as the membranes in some simulations, in order to have proper thermal boundary conditions, were over 1 mm^2. Additionally, nanophotonic light absorption occurred on a much smaller time scale than the thermal processes, so the nanophotonic absorption could be treated as a steady-state power input into the thermal simulations. We assumed that the optical and other properties of materials did not change significantly over the small temperature ranges in our simulations, although if larger temperatures are generated or materials with significantly temperature-dependent optical properties are used, it would be necessary to incorporate these temperature-dependent changes into the model.

2.3 Thermal time constant of devices

The thermal time constant of nanophotonic devices depends primarily on the size of the structure being heated as well as the thermal conductivity of the material and its surroundings. The thermal time constant, τ, for convective cooling or heating of a simple geometry can be taken as

$$\tau = \frac{\rho c_p V}{h A_s}, \tag{2.1}$$

where ρ is density, c_p is heat capacity, V is volume, h is the heat transfer coefficient between the object and its surroundings, and A_s is the surface area. If examining conductive instead of convective heating or cooling, the thermal conductivity would replace h to first approximation in the denominator of the time constant equation, as it plays a similar role. We can see that the larger structures will have a larger time constant, while higher h (or thermal conductivity) will decrease the time constant. Therefore, a small, high thermal conductivity structure will have fast response time. On the other hand, high thermal conductivity will decrease the maximum temperature achievable by a nanophotonic thermoelectric device under illumination, which will decrease responsivity (sensitivity). Therefore, there is a

tradeoff between responsivity and response time which must be tailored for the specific application.

2.4 Simulation material values

Starting with nanophotonic (EM) and thermal simulations, we can determine the performance characteristics of thermoelectric nanophotonic devices. In the sections below, several different nanophotonic designs coupled to different thermal management designs will be explored.

In this chapter, the following values are used in simulations. Heat capacity of Bi_2Te_3 (Gorbachuk and Sidorko, 2004) was taken as 158 J/kg·K, density (Nolas, Sharp, and Goldsmid, 2001) as 7,859 kg/m^3, surface emissivity as 0.34, and thermal conductivity (D. M. Rowe and Bhandari, 1995) as 2.05 W/m·K. SiN_x heat capacity (Greenwood and Earnshaw, 1984) was taken as 800 J/kg·K, density as 3,185 kg/m^3, surface emissivity as 0.9, and thermal conductivity (Ftouni et al., 2015) as 2.5 W/m·K. SiO_2 heat capacity (Greenwood and Earnshaw, 1984) was taken as 703 J/kg·K, density as 2,196 kg/m^3, surface emissivity as 0.9, and thermal conductivity (Yamane et al., 2002) as 1.38 W/m·K. Au heat capacity was taken as 129 J/kg·K, density as 19300 kg/m^3, surface emissivity as 0.025, and thermal conductivity as 317 W/m·K.

2.5 Perfect absorber

Nanophotonic design

To increase the sensitivity of a detector, it is desirable to absorb close to 100% of the incident light. While a carbon black absorber can do this, they are often large (10s or 100s of microns thick) and thus, slow due to their large heat capacity. Nanophotonic structures can be used to make close to 100% absorbing structures within a wavelength range, with a much smaller footprint than carbon black. The perfect absorber design we explored was modeled off a design from the literature (Liu et al., 2010a), and is shown artistically in Figure 2.1. An additional benefit of these structures is the insensitivity to incident angle of illumination, as will be shown. The cylinder height was 20 nm, the radius 176 nm, and the pitch 600 nm. The dielectric gap was 30 nm thick SiO_2 with index from Palik (Palik, 1997a). The backreflector was 50 nm gold with index data from Johnson and Christy (P. B. Johnson and Christy, 1972), beneath which was 100 nm of bismuth telluride (see Figure 2.2 for measured dielectric function). The magnitude of the electric field at the resonant wavelength (at peak absorption) of 1,648 nm is shown in Figure

2.6 (left). As a function of wavelength, the absorption of the structure is shown in Figure 2.6 (right).

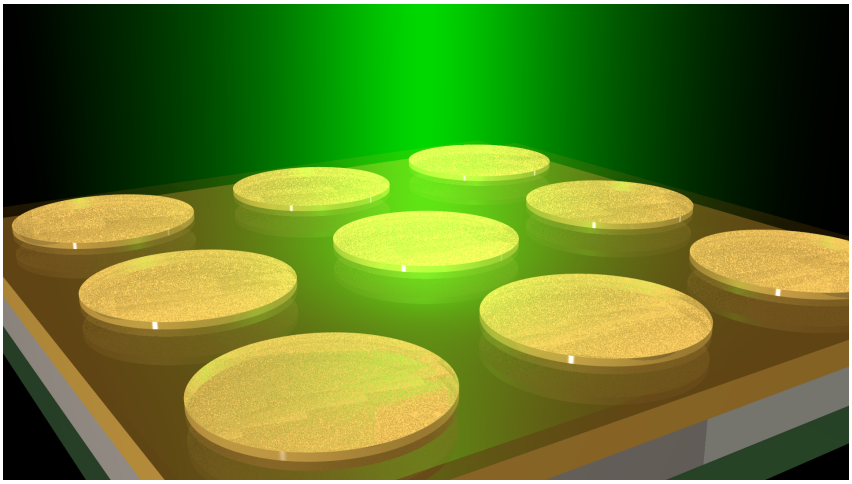

Figure 2.1: Artistic rendition of perfect absorber structure, rendered in POV-ray.

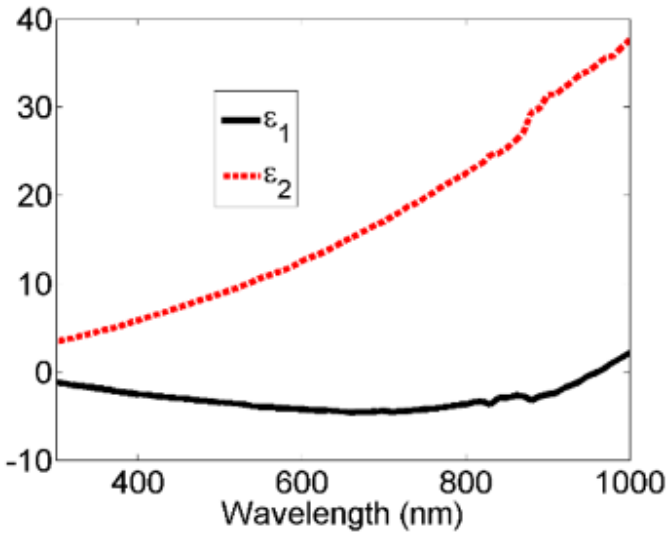

Figure 2.2: Dielectric function of bismuth telluride in visible and near-IR wavelengths measured using J.A. Woollam Co. VASE and IR-VASE MARK II ellipsometers and analyzed with WVASE software..

Thermal design

In the thermal simulations, a patch of the perfect absorber materials was placed in the center of the 100 nm thick thermoelectric junction on top of a 10 μm wide, 100 nm thick SiN_x membrane suspended between two large blocks of SiO_2, shown in Supplementary Fig. 12a and Supplementary Fig. 10j. Sandwiched between the

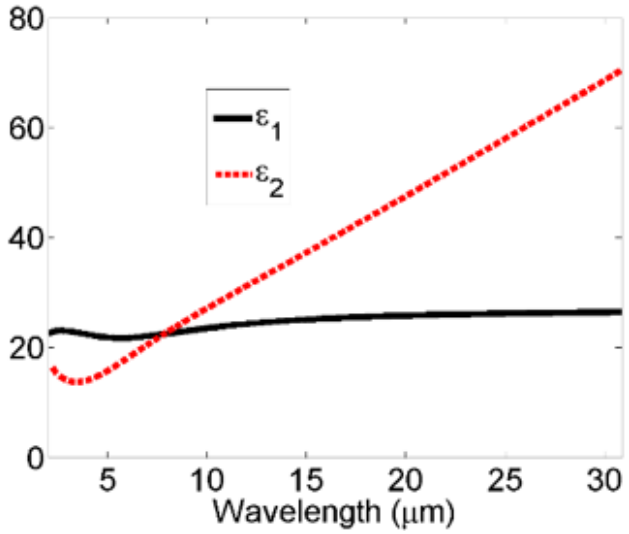

Figure 2.3: Dielectric function of bismuth telluride in IR wavelengths.

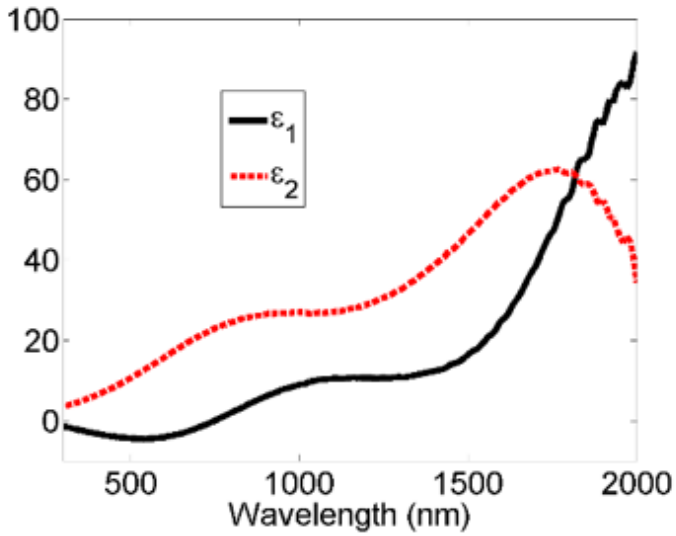

Figure 2.4: Dielectric function of antimony telluride in visible to near-IR wavelengths.

large SiO_2 blocks and the "cold" end of the SiN_x membrane is a 200 nm thick layer of Au. Because Au has a large thermal conductivity, this layer acts as a heat sink for the "cold" end of the thermoelectric materials, enhancing the temperature gradient across the thermoelectric materials. The simulations neglected convection, i.e. were simulated a vacuum environment.

Using these thermal simulations, we can plot the difference in temperature between

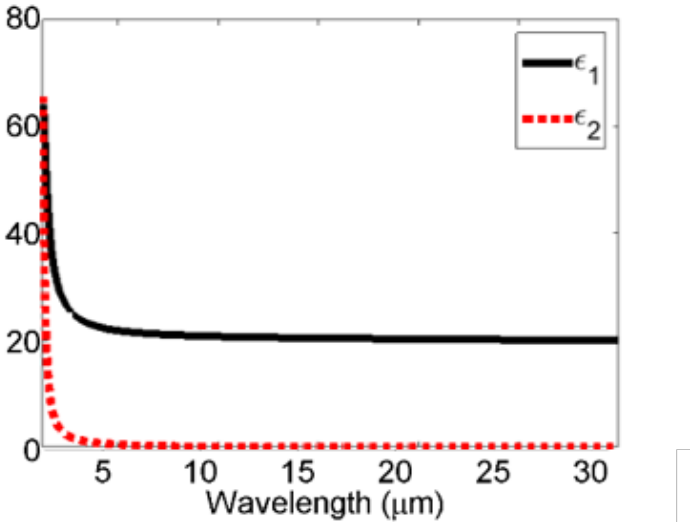

Figure 2.5: Dielectric function of antimony telluride in IR wavelengths.

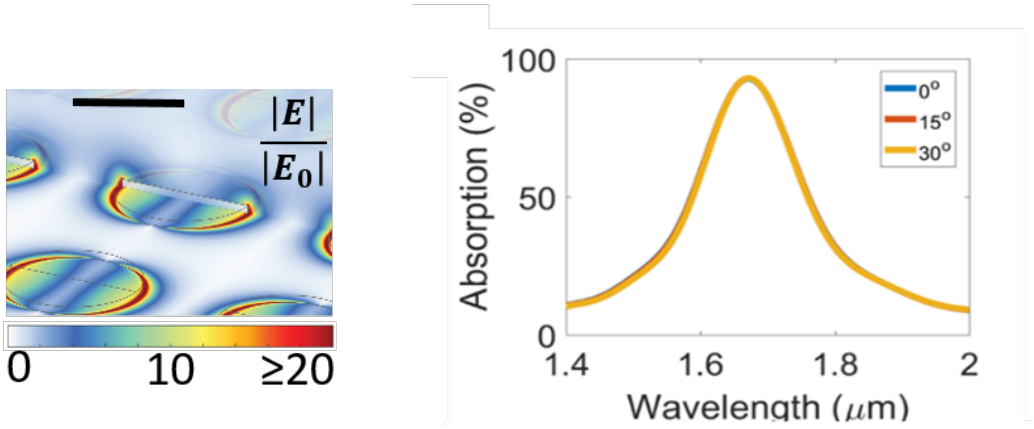

Figure 2.6: Lumerical simulation of normalized electric field magnitude of the perfect absorber structure described in the text, on resonance at 1,648 nm (left), and as a function of wavelength (right). Scale bar is 300 nm. Credit to Dr. Seyoon Kim for nicely plotting the simulated data on the left.

the hot and cold edges of the bismuth telluride material as a function of power absorbed by the perfect absorber structure, as shown in Figure 2.8. The simulated responsivity as a function of wavelength is shown in Figure 2.9, and Figure 2.10 shows noise equivalent power as a function of wavelength for this structure. Noise equivalent power is caluculated by dividing the simulated responsivity by the theoretical Johnson noise of the structure at room temperature, using the geometry and measured resistivity of our materials to compute resistance.

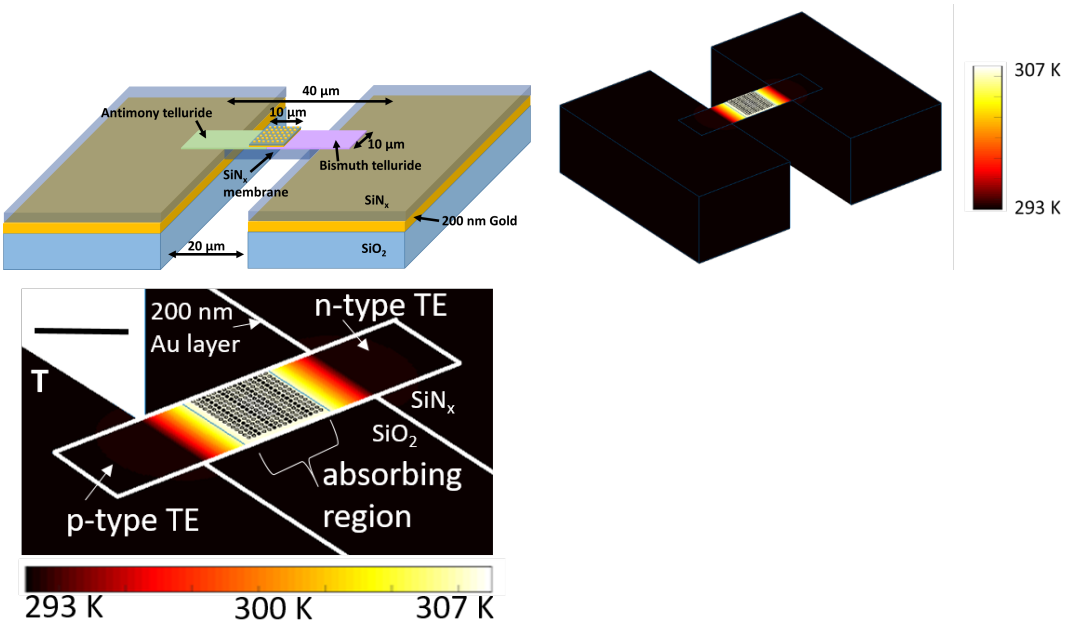

Figure 2.7: Perfect absorber and split ring resonator thermal simulation design. (Top left) Diagram of thermal simulation design for a perfect absorber and a split ring resonator thermal simulation corresponding to data in Figures 2.9, 2.15 and Figures 2.8, 2.14, 2.10, 2.16, 2.13. The split ring resonator design has a 50 nm thick SiN_x membrane with 60 nm of thermoelectric material above it, which serves as a backreflector in the optical design. The perfect absorber has a 100 nm thick SiN_x membrane and 100 nm thick TE materials. It uses a 50 nm thick layer of gold as the backreflector in the 10 µm by 10 µm center absorber patch. The 200 nm gold above the silica increases thermal conduction of heat from the cold end of the device (i.e. acts as a heat sink), and any other thermally conductive material would serve this purpose well. Thermal simulations involved a volumetric heat influx into the centrally located split ring resonator or cylinder absorbers in the array (this assumes absorption near the edges of the arrays would be worse). (Top right) Thermal simulation of perfect absorber with a total absorbed power of 23 µW. The split ring resonator thermal profile was similar (see Figure 2.13). (Bottom left) Thermal simulations of the perfect absorber design. The perfect absorber array covers a 10 µm by 10 µm square in the center of a suspended, TE junction on top of a SiN_x membrane. The "cold" ends of the TE materials sit on a 100 nm SiN_x /200 nm Au/20 µm SiO_2 substrate. The Au layer acts as a high thermal conductivity heat sink to conduct heat away from the "cold" ends of the TE materials. Both simulations were performed in vacuum. The scale bars are 10 µm, and the absorbed power is 23 µW.

2.6 Split ring resonator perfect absorber

Nanophotonic design

The split ring resonator can also be a perfect absorber, angle-insensitive design if the backreflector is a low-loss metal in the wavelength range of interest (such as

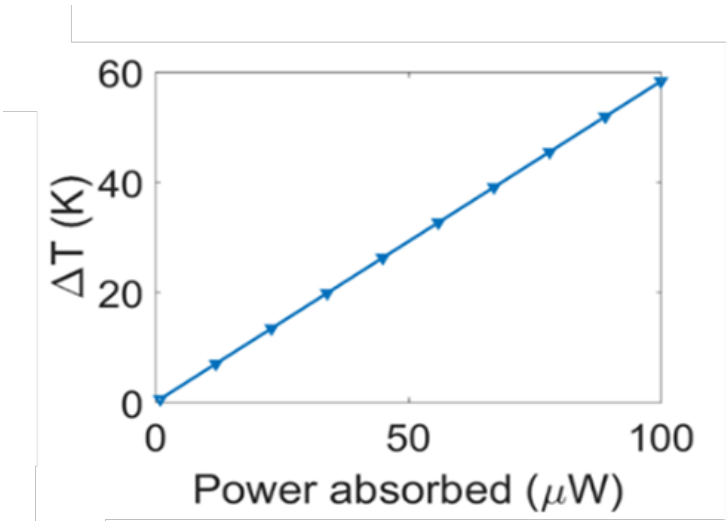

Figure 2.8: Temperature differences between the hot and cold sides of the thermoelectric materials as a function of power absorbed for the perfect absorber structure.

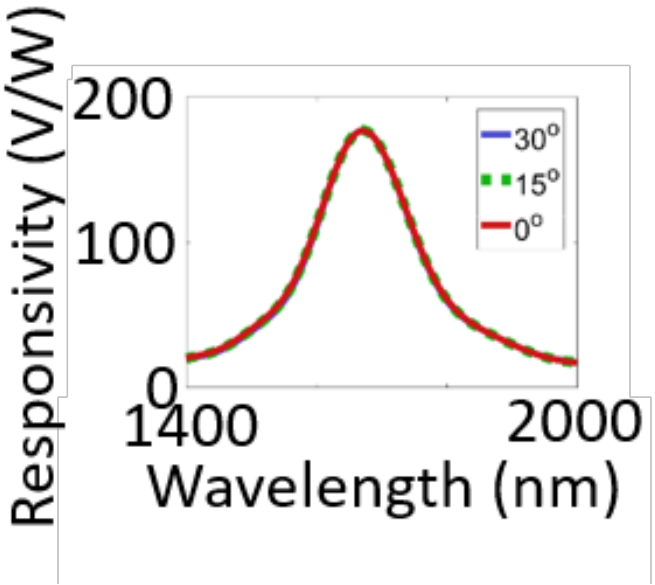

Figure 2.9: Simulated responsivities as a function of incident angle of the perfect absorber structure.

gold in the near-IR). The split ring resonator perfect absorber (shown artistically in Figure 2.11) consists of two gold rings, the outer ring with outer radius 70 nm and inner radius 60 nm, with a split 15 nm wide. The inner ring had outer radius of 40 nm and inner radius of 30 nm, with split 20 nm wide. Both rings were 20 nm tall and had optical constants from Johnson and Christy (P. B. Johnson and Christy,

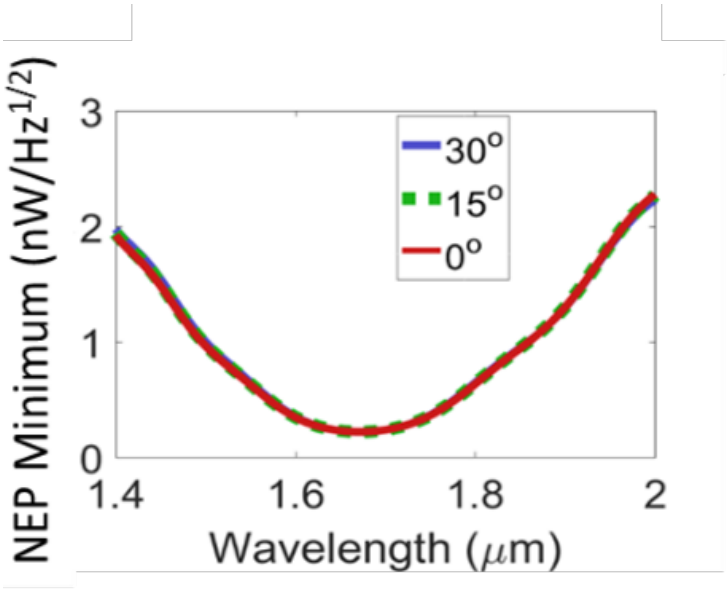

Figure 2.10: Simulated noise equivalent power as a function of incident angle of the perfect absorber structure.

1972). From center to center, the distance between each split ring resonator in the array is 240 nm. The dielectric layer was 40 nm of Palik (Palik, 1997a) SiO_2. The backreflector layer was made of antimony telluride. The thermal simulations were performed in a similar manner as the perfect absorber structures described above, but with a 60 nm thick thermoelectric layer and 50 nm thick SiN_x membrane below.

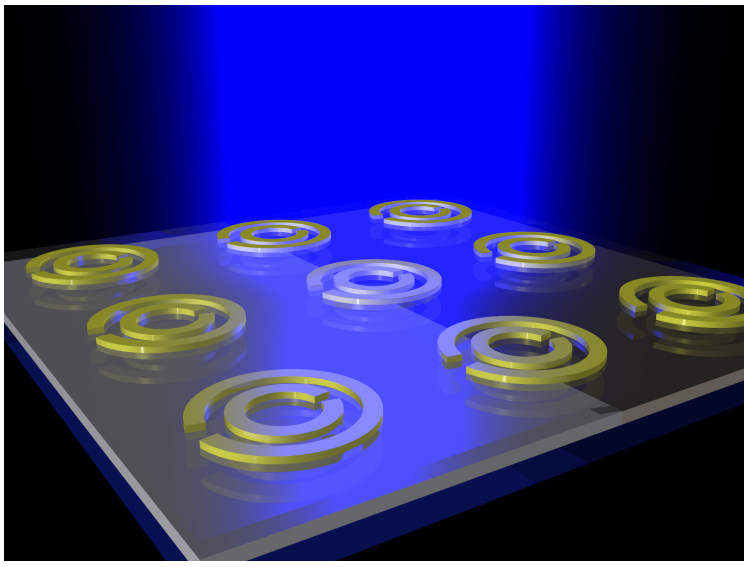

Figure 2.11: Artistic rendition of split ring resonator design, rendered in POV-ray.

The magnitude of the electric field at one resonance can be found in Figure 2.12

(left), and the absorption as a function of wavelength and incident angle can be found in Figure 2.12 (right).

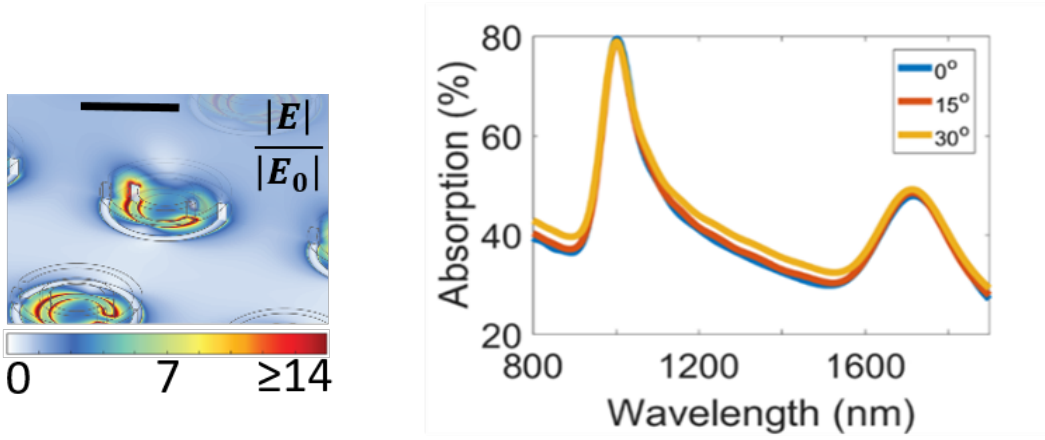

Figure 2.12: Lumerical simulation of normalized electric field magnitude of the split ring resonator structure described in the text, on resonance at 1,710 nm (left), and as a function of wavelength (right). Scale bar is 50 nm. Credit to Dr. Seyoon Kim for nicely plotting the simulated data on the left.

Thermal design

The thermal design for this structure is similar to the perfect absorber structure above, but with a thinner membrane, which leads to a greater temperature difference, shown in Figure 2.13. We can plot the temperature difference between the hot and cold ends of the thermoelectric elements in Figure 2.14. The simulated responsivity is shown in Figure 2.15, and the simulated noise equivalent power is shown in Figure 2.16.

2.7 Plasmonic bowtie antenna

Nanophotonic design

Plasmonic structures can focus certain light wavelengths into mode volumes much smaller than their free-space wavelength. One example is the bowtie structure artistically rendered in Figure 2.17. The bowtie structure simulated had an internal angle of 90 degrees and was cut from a circle of radius 500 nm and thickness 80 nm. The gap between the "wings" of the bowtie was 30 nm. The wire was 20 nm wide and 20 nm tall and made of one material, bismuth telluride, due to memory and symmetry constraints. The bowtie was gold with index of refraction from Palik

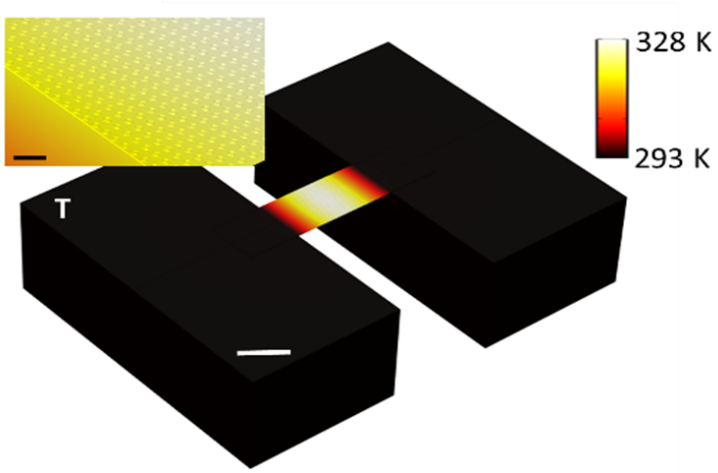

Figure 2.13: Thermal simulation for split ring resonator perfect absorbers utilizing a thin membrane to lower thermal heat loss to the substrate. Total absorbed power is 23 µW. Large scale bar is 10 µm.

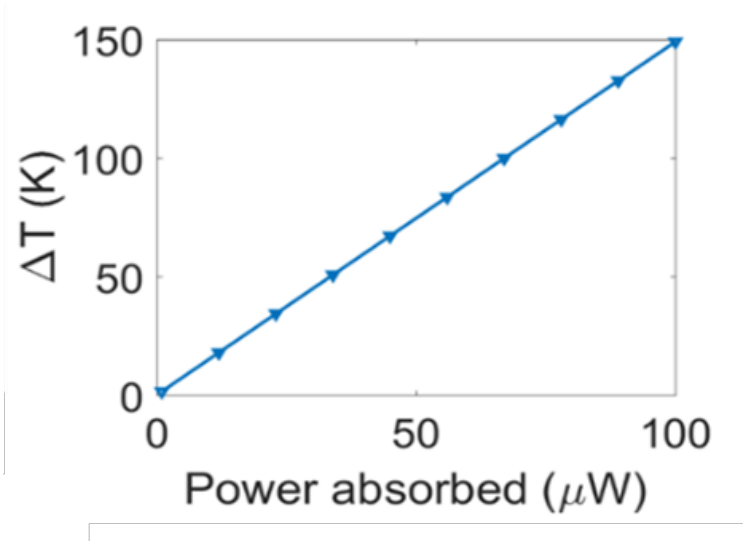

Figure 2.14: Temperature differences between the hot and cold sides of the thermoelectric materials as a function of power absorbed for the split ring resonator absorber structure.

(Palik, 1997a). The substrate was an infinitely thick slab of SiO_2 with index of 1.455.

The magnitude of the electric field at one resonance can be found in Figure 2.18 (left), and the absorption cross section as a function of wavelength and incident angle can be found in Figure 2.18 (right).

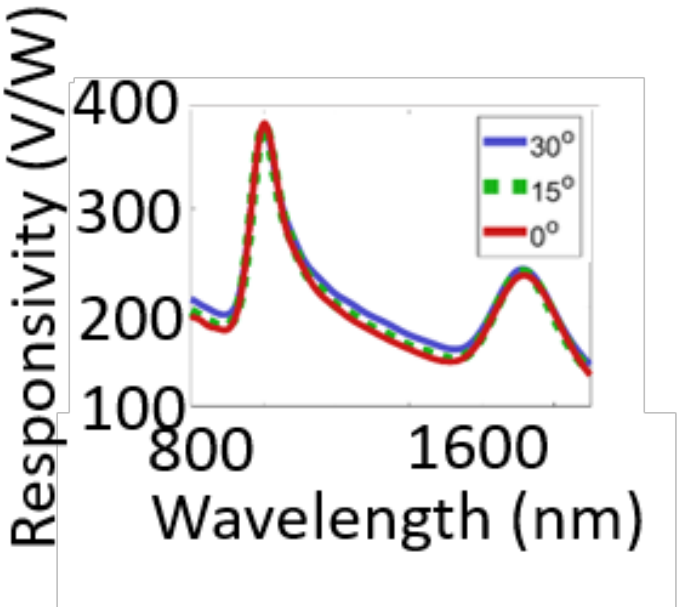

Figure 2.15: Simulated responsivities as a function of incident angle of the split ring resonator absorber structure.

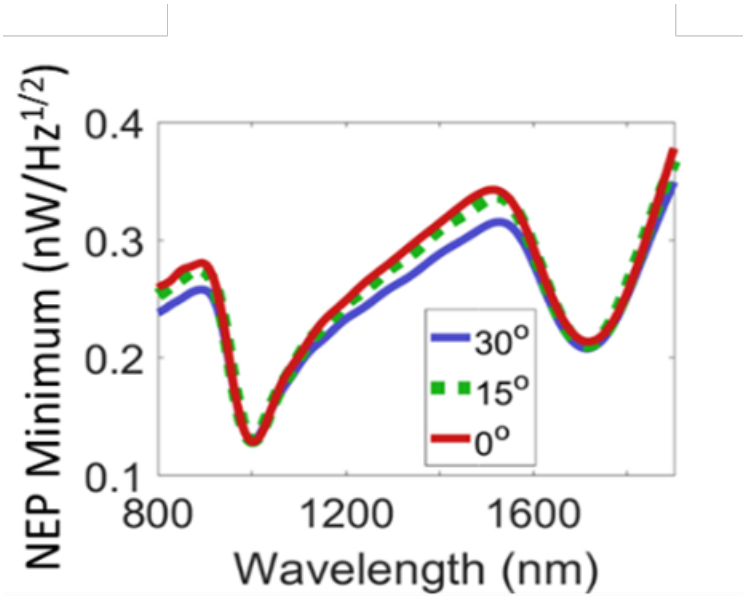

Figure 2.16: Simulated noise equivalent power as a function of incident angle of the split ring resonator absorber structure.

Thermal design

The thermal design for this structure consisted of placing a 20 by 20 nm thermo-electric wire junction at the plasmonic focusing point of the bowtie antenna. A

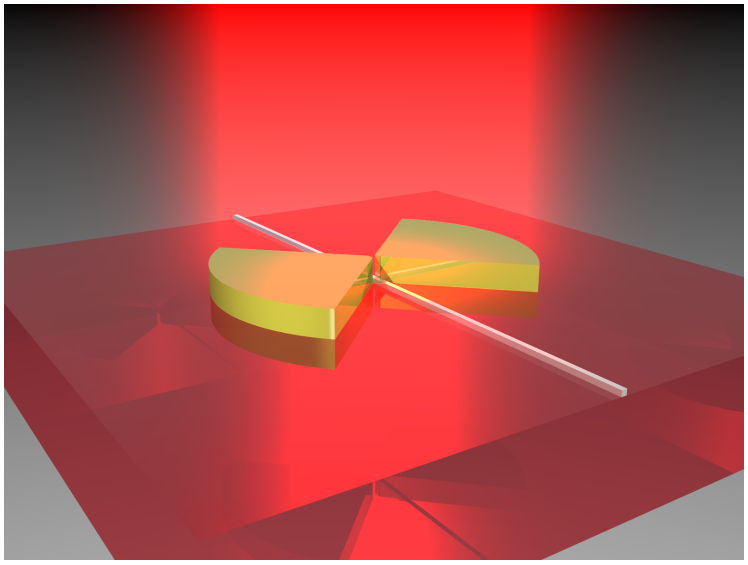

Figure 2.17: Artistic rendition of the plasmonic bowtie absorber, rendered in POV-ray.

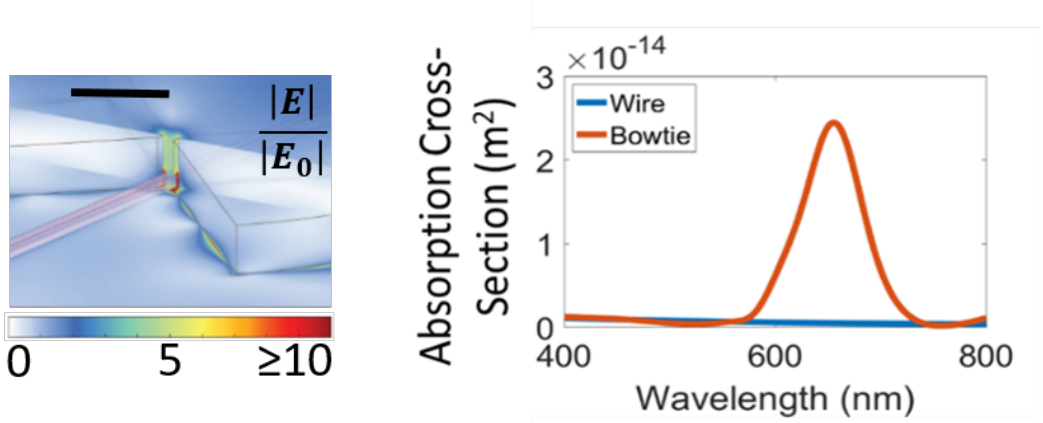

Figure 2.18: Lumerical simulation of normalized electric field magnitude of the bowtie absorber structure described in the text, on resonance at 660 nm (left), and absorption cross-section as a function of wavelength (right) for the wire alone (blue) and the wire with the bowtie (orange). Scale bar is 200 nm. Credit to Dr. Seyoon Kim for these simulation results and for nicely plotting the simulated data on the left.

COMSOL simulation of the bowtie is shown with several different temperature scales in Figure 2.19.

The temperature difference between the hot spot at the center of the bowtie antenna and the edge of the thermoelectric wires is shown in Figure 2.20. Figure 2.21 shows the responsivity of this structure, and 2.22 shows the noise equivalent power.

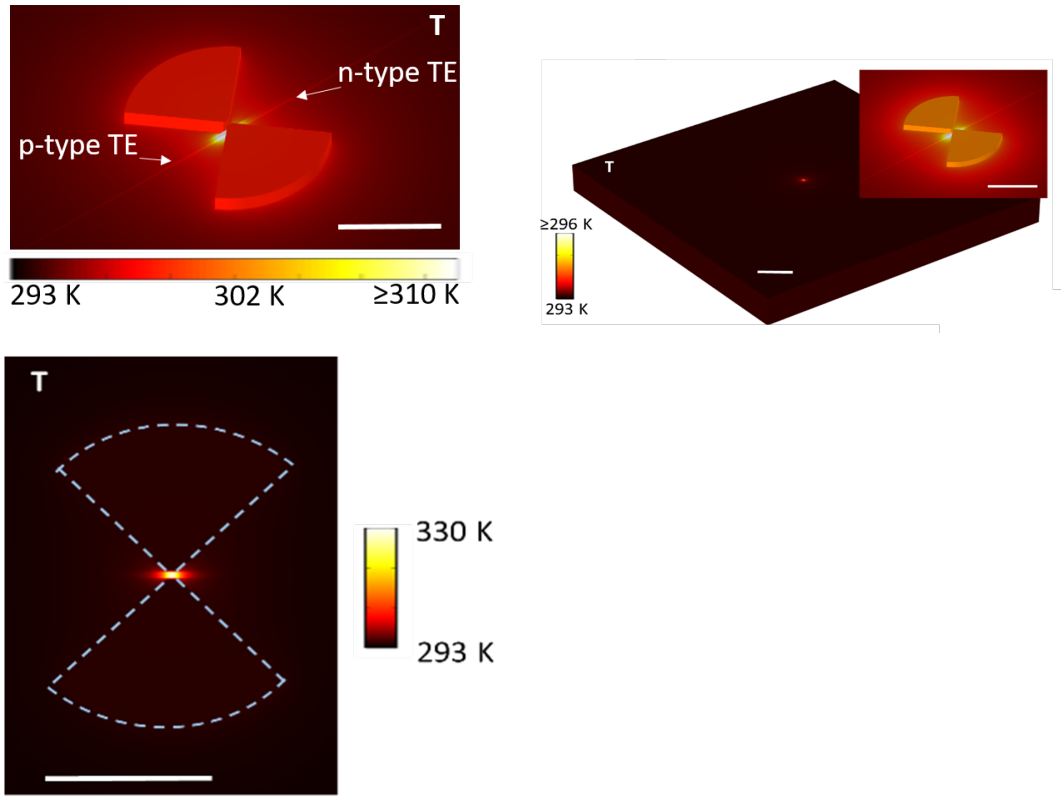

Figure 2.19: (Top left) Thermal simulations of teh resonant bowtie antenna, performed in vacuum. Scale bar is 500 nm and the absorbed power is 7.9 µW. (Top right) Thermal simulation of bowtie structure with absorbed power of 7.9 µW. Maximum temperature is 330 K, but a smaller scale was used to show temperature differences better. Scale bar is 20 µm and inset scale bar is 500 nm. (Bottom left) The same thermal simulation as in the top right is shown, but with the full temperature scale bar. The outline of the bowtie is given in blue dotted line. Scale bar is 500 nm.

2.8 Guided mode resonator thermocouple

The guided mode resonance thermocouple explored in this section was fabricated and experimentally tested in the next chapter. Here we present an in-depth analysis of the nanophotonic structure.

Nanophotonic design

This section is broken up into two parts. The first is concerned solely with why the given guided mode resonance structure absorbs the wavelength of light it does, and the second part looks at how this guided mode resonance design can be incorported with thermoelectrics, to generate a necessary temperature gradient. An artistic rendition of the thermocouple structure is shown in Figure 2.23. Figure 2.23 shows

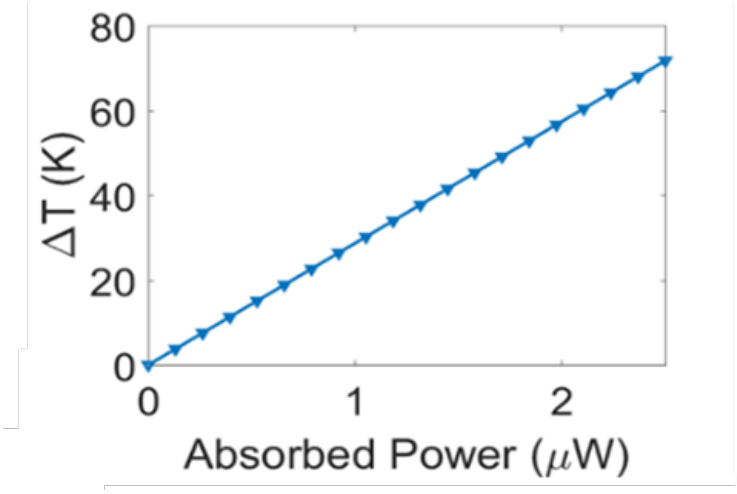

Figure 2.20: Temperature difference between hot and cold spot of thermoelectric wires in the resonant bowtie antenna design.

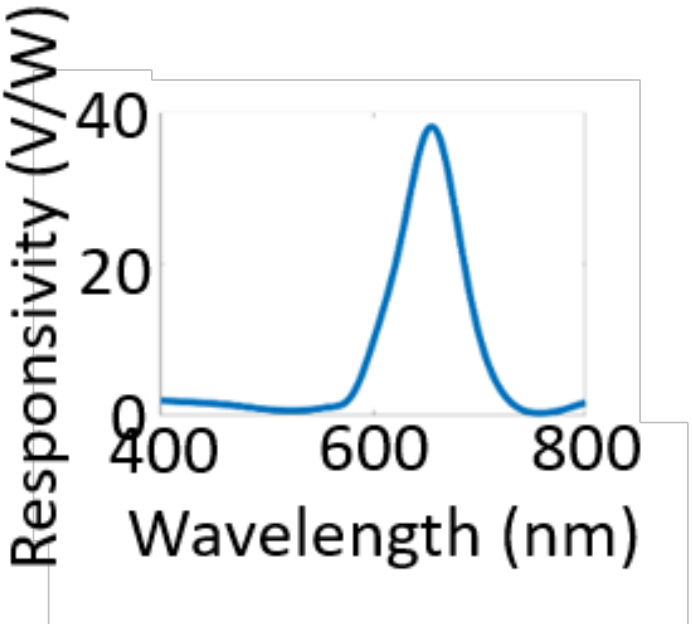

Figure 2.21: Responsivity of the resonant bowtie antenna design.

a schematic of our experimental structure, a guided mode resonance wire array, with wire dimensions of 40 nm × 100 nm × 50 μm, in which TM polarized, normal incident, unfocused optical radiation is coupled into a waveguide mode via a periodic thermoelectric wire array that serves as a light absorber.

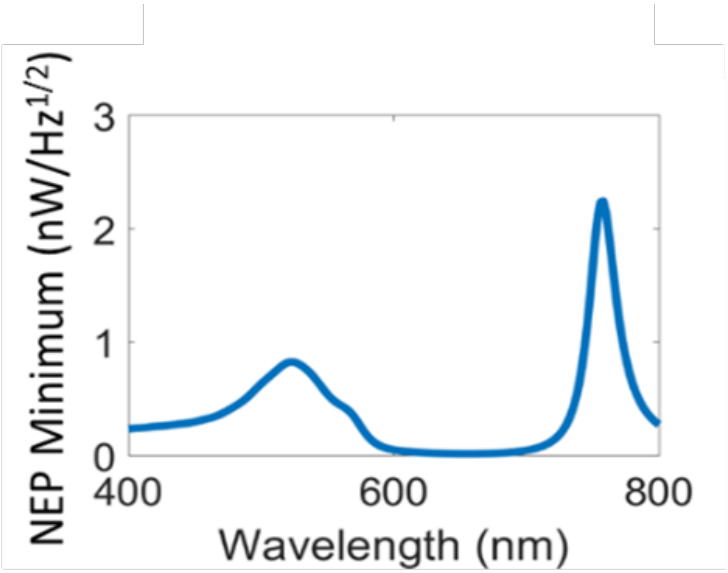

Figure 2.22: Noise equivalent power of the resonant bowtie antenna design.

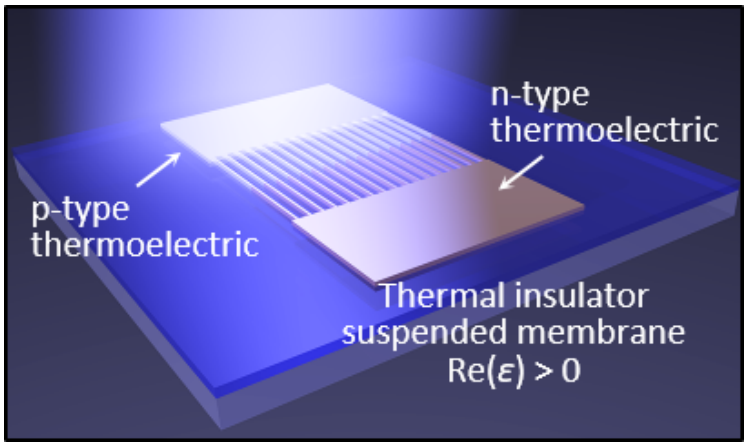

Figure 2.23: Conceptual design of GMR resonant thermoelectric structure. The junction of the thermoelectric wires absorbs incident, spatially uniform illumination and is absorbed, heating the junction and producing a thermoelectric voltage. Light illuminating the pads will be reflected, creating the necessary temperature gradient.

Guided mode resonance with Fano analysis

Nanophotonic thermoelectric structures must concentrate the electric field in the thermoelectric material to maximize absorption. Our guided mode resonance structures achieve this via Fano interference (Fano, 1961) of a waveguide mode and a Fabry-Perot resonance in the waveguide, which will be described in detail. The resonant wavelength of this waveguide mode is predicted quite well by the grating coupler equation for normally incident light, assuming infinitely narrow gratings,

$2\pi/d = \beta$, where d is the grating pitch and β is the propagation constant of the two-layer slab waveguide. Small deviations from the grating coupler equation predictions are due to waveguide mode interactions with Fabry-Perot resonances.

Fano lineshapes are produced when a continuum of states interacts with discrete or narrow modes near the same energy, and appear in electronic circuits, nanophotonics, and atomic spectra (Gallinet, 2012). In our specific case, we have a broad, Fabry-Perot resonance in our waveguide layers acting as the continuum background (radiative bright mode), and a narrow, waveguide mode (nonradiative dark mode) interacting with it. The effect is developed thoroughly in work by Gallinet et al. (Gallinet and O. J. F. Martin, 2011a; Gallinet, 2012; Gallinet and O. J. F. Martin, 2011b), which will be summarized here.

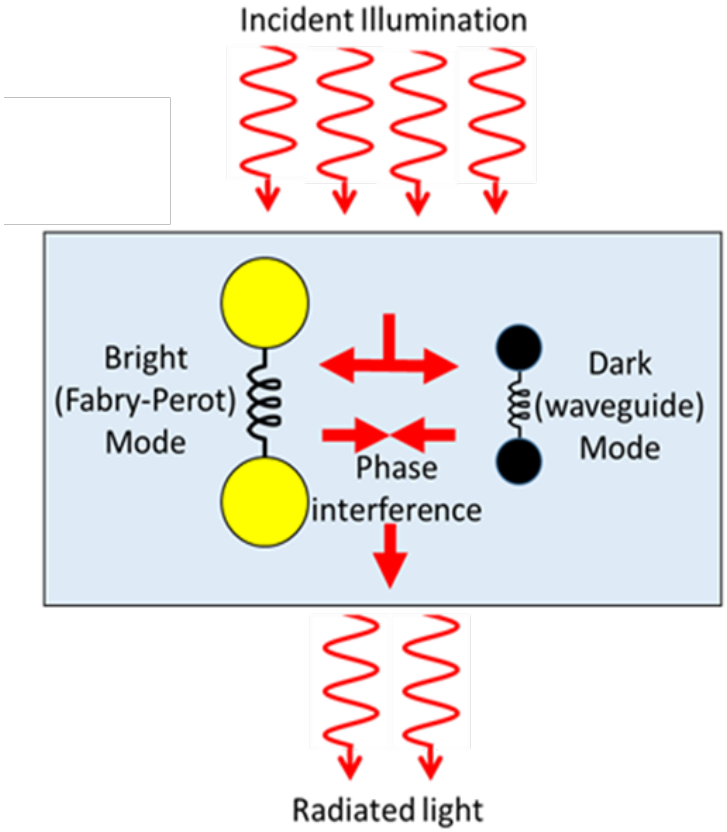

Figure 2.24: Diagram of interaction of bright (broad resonance) and dark (narrow resonance) modes in the production of Fano lineshapes. The bright mode is the Fabry-Perot resonance of light with k-vector parallel to incident illumination. Analysis from (Gallinet and O. J. F. Martin, 2011a).

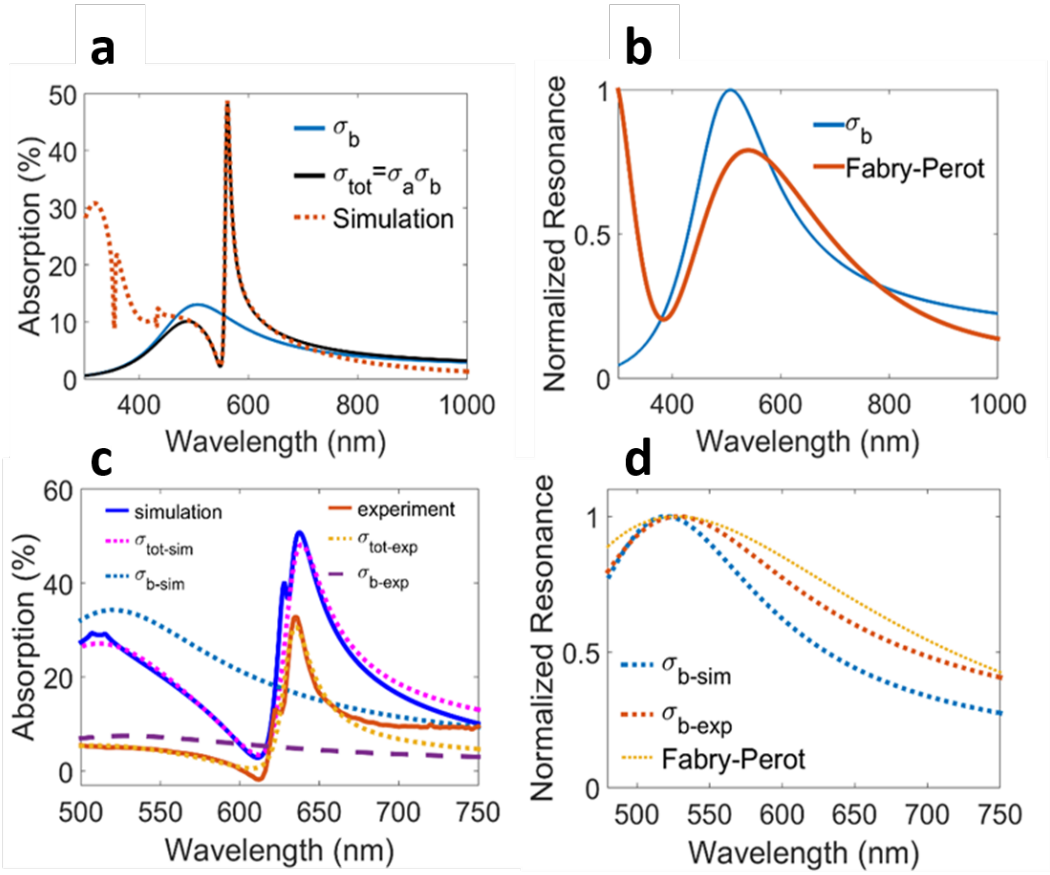

Figure 2.25: **a**, The Fano formula (Equation 2.7) is fit to our simulation data for a region near the resonance. textbfb, The extracted bright mode from **a** is shown in more detail, and is compared with the normalized magnitude of the electric field at the top of a 50 nm of SiO_2/100 nm of SiN_x waveguide without wires due to Fabry-Perot resonances. **c**, Bright mode and Fano formula fits to simulated and experimental data. **d** The simulation and experimental bright modes from **c** along with the Fabry-Perot resonance for a waveguide-only structure.

From the interaction between the bright (continuum) and dark (waveguide) modes, as Figure 2.24a outlines, we get a new resonance of the combined system, at a position equal to

$$\omega_a = \omega_d^2 + \omega_d \Delta, \tag{2.2}$$

where ω_d is the resonant frequency of the dark mode and Δ is the shift away from this frequency due to coupling with the bright mode calculated explicitly in Gallinet

(Gallinet, 2012). The shape of the new resonance is asymmetric about the new resonance, ω_a^2 , and is given by

$$\sigma_a(\omega) = \frac{\left(\frac{\omega^2-\omega_a^2}{\Gamma} + q\right)^2 + b}{\left(\frac{\omega^2-\omega_a^2}{\Gamma}\right)^2 + 1}, \tag{2.3}$$

where q is an asymmetry term, Γ is a width term equal to $2\omega_a W_a$ (W_a is a width) for $\omega_a >> W_a$, and b is a modulated damping term (Gallinet, 2012). The asymmetry of this dark mode comes from a rapid phase asymmetry of π across the resonance from the original dark mode, interfering with the symmetric phase difference in the bright mode across the resonance. On one side of the resonance, these bright and dark modes destructively interfere, and on the other side of the resonance they constructively interfere. The location of destructive interference on either side of the resonance depends on the sign of the phase difference between the dark and bright mode resonances, along with whether the loss is real or imaginary. This is expressed in the asymmetry term in Equation 2.3, q, where

$$q = \pm\frac{\left(\omega_d^2 - \omega_b^2\right)}{2\Gamma_b\left(1 + \frac{\Gamma_i}{\Gamma_c}\right)}, \tag{2.4}$$

where Γ_b is equal to $2\omega_b W_b$ (W_b is a width) for $\omega_b >> W_b$, Γ_i is intrinsic loss, and Γ_c is coupling loss. b, the modulation damping parameter, is equal to

$$b = \frac{\left(\frac{\Gamma_i}{\Gamma_c}\right)^2}{\left(1 + \frac{\Gamma_i}{\Gamma_c}\right)^2} = \left(\frac{\Gamma_i}{\Gamma_c + \Gamma_i}\right)^2. \tag{2.5}$$

The bright mode Fano formula, on the other hand, is pseudo-Lorentzian and therefore symmetric, given by

$$\sigma_b(\omega) = \frac{a^2}{\left(\frac{\omega^2-\omega_b^2}{\Gamma_b}\right)^2 + 1}. \tag{2.6}$$

From Gallinet (Gallinet, 2012), we find that the total optical response of the system comes from multiplying the Fano formulas of the bright mode with the dark mode modified by the bright mode, or

$$\sigma_{tot}(\omega) = \sigma_b(\omega)\sigma_a(\omega). \tag{2.7}$$

Fitting our resonance to this formula, the frequency of the bright mode resonance can be extracted to determine its source, whether it be from plasma resonances of the wires, incident radiation, or Fabry-Perot resonances in the waveguide. Figure 2.25a shows a full wave simulation of Sb_2Te_3 wires that are 60 nm wide, 40 nm thick, with a pitch of 431 nm on a 50 nm SiO_2/100 nm SiN_x suspended waveguide, that is fit to the Fano formula (Equation 2.7). Shown are the simulation (dotted orange), the fit of the combined Fano formula (black, Equation 2.7) between 440 nm and 650 nm, and the extracted Fano formula of the bright mode (blue, Equation 2.6). All fitting parameters for Figures 2.25a-d are located in Table 2.1.

The extracted bright mode from Figure 2.25a is shown in more detail in Figure 2.25b, and is compared with the normalized magnitude of the electric field at the top of a 50 nm of SiO_2/100 nm of SiN_x waveguide without wires due to Fabry-Perot resonances. The Fabry-Perot resonance (orange) is calculated via full-wave simulation as the normalized magnitude of the electric field at a point on the surface of a bare waveguide structure. The bright mode fit (blue) is described in Figure 2.25a. The shift in the Fabry-Perot peak is associated with the contributions of the wires to the effective index of the entire photonic crystal structure. The minor misalignment of the bright mode and Fabry-Perot peak is likely caused by the grating itself altering the location of the bright mode, as the effective index of the photonic crystal made up of the grating plus the waveguide will be different than the index of the waveguide alone.

Using this method to compare our experimental and simulated data, we can compare differences quantitatively. Figure 2.25c shows such a comparison: the experiment versus simulation for 45 nm SiO_2/100 nm SiN_x suspended waveguide with Sb_2Te_3 wires. Wire thickness is 40 nm, width is 89 nm, and pitch is 511 nm. Shown are the full wave simulation for this structure (solid blue), the total Fano function fit (dotted magenta) to the full wave simulation with parameters given in Table 2.1, and the bright mode (dotted blue) extracted from the total Fano fit function, corresponding to the Fabry-Perot resonance. Also shown are the measured absorption (solid orange),

the fitted total Fano function to the measured absorption (dotted yellow), and the extracted bright mode profile from the measured extinction (dashed purple). Figure 2.25d plots the simulation and experimental bright modes from Figure 2.25c along with the Fabry-Perot resonance for a waveguide-only structure. Extracted bright mode for the experimental (dotted orange) and simulated (dotted blue) data from 2.25c, with the simulated electric field magnitude (dotted yellow) at the surface of the waveguide (without the wires) due to the Fabry-Perot resonance. All curves are normalized to their maximum value in the given wavelength range for ease of comparing resonance peaks. The experimental, simulated, and Fabry-Perot peaks align reasonably well.

If we compare the various values of the fitting parameters from Table 2.1, we note that b, the modulated damping term, is higher in the experiment than in the simulation. From Equation 2.5, we can see that this indicates the intrinsic loss has a larger influence (or the coupling loss has a lesser influence) in the experiment than in the simulation. This could be attributed to a, the bright mode amplitude, having a higher magnitude in the simulation than in the experiment.

Figure	a (unitless)	ω_a (eV)	W_a (eV)	ω_b (eV)	W_b (eV)	q (unitless)	b (unitless)
Figures 2.25a,b	0.13038± 0.000704	2.2174± 0.00014	0.020862± 0.000133	2.4448± 0.0067	0.48832± 0.00726	-1.6986± 0.00943	0.82025± 0.0388
Figures 2.25c,d simulation	0.58557± 0.00514	1.9725± 0.000991	0.041887± 0.000941	2.383± 0.0233	0.38067± 0.0203	-1.2762± 0.0227	0.53022± 0.0578
Figures 2.25c,d experiment	0.27396± 0.00968	1.9637± 0.00144	0.032595± 0.00151	2.3473± 0.0563	0.48954± 0.0991	-2.185± 0.11	0.6564± 0.412

Table 2.1: Fitting parameters for Equation 2.7 for Figure 2.25. 95% confidence intervals are given.

Absorption curve	Pitch$_s$ (nm)	Width$_s$ (nm)	Θ_s (deg)	Pitch$_f$ (nm)	Width$_f$ (nm)	Θ_f (deg)	Scaling (unitless)
i	567	97	0.5	560	97	0.5	0.71
ii	566	91	0.5	560	90	0.5	0.66
iii	511	119	0.5	507	102	0.6	1.06
iv	509	98	0.5	507	87	0.6	0.87
v	511	89	0.5	507	82	0.6	0.76
vi	452	131	1	455	102	1	0.99
vii	452	101	1	455	87	1	0.87

Table 2.2: Comparison of experimental dimensions and illumination angle (Pitch$_s$, Width$_s$, Θ_s) with best-fit simulation dimensions, illumination angle (Pitch$_f$, Width$_f$, Θ_f) and scaling factor corresponding to Figure 2.29 in the main text.

A wide range of materials with varying Seebeck coefficients including Al, Cr, and Sb_2Te_3 give rise to guided mode resonances with very similar peak heights, positions, and widths, as shown in Figure 2.26. Sb_2Te_3 and Cr exhibit a large

extinction coefficient k at the waveguide resonance wavelength and are plasmonic (ϵ'<0) in this wavelength range. By contrast Al has a more negative value of ϵ' in this region and has a narrower resonant linewidth, whereas Au and Cu have resonances that are spectrally shifted in wavelength due to interband transitions or plasmon resonances that couple to the waveguide mode, causing a Rabi splitting of the modes (Christ et al., 2004).

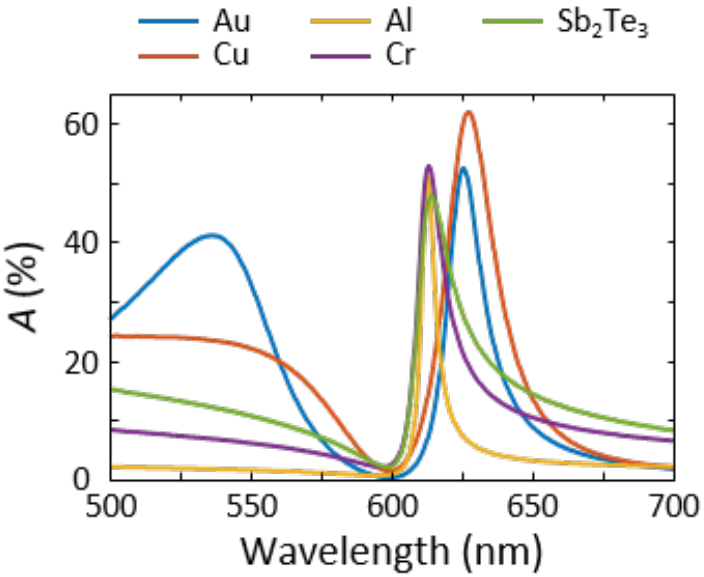

Figure 2.26: A comparison of absorption spectra of different wire materials in our guided mode resonant structure composed of 40 nm high, 68 nm wide, pitch of 488 nm wires on a waveguide of 50 nm SiO_2/100 nm SiN_x.

Cross-sections of Sb_2Te_3 wire guided mode resonance structures are shown in Figure 2.27. Figure 2.27a,c correspond to the absorption maximum wavelength, and Figure 2.27b,d correspond to the absorption minimum just to the left of the maximum, as shown in Fig. 2a (Sb_2Te_3). For this geometry, the absorption maximum occurs at 611 nm, and the absorption minimum occurs at 596 nm. Figure 2.27a shows the electric field surrounding the wires at the maximum absorption wavelength, resulting from a constructive interference of the waveguide mode and the Fabry-Perot resonance. The large electric field magnitude in the wire corresponds to high power absorption on resonance, shown in Figure 2.27c, whereas Figure 2.27b illustrates the off-resonance electric field, at an absorption minimum, shown in Figure 2.27d.

Thermoelectric nanophotonic structures supporting guided mode resonances exhibit

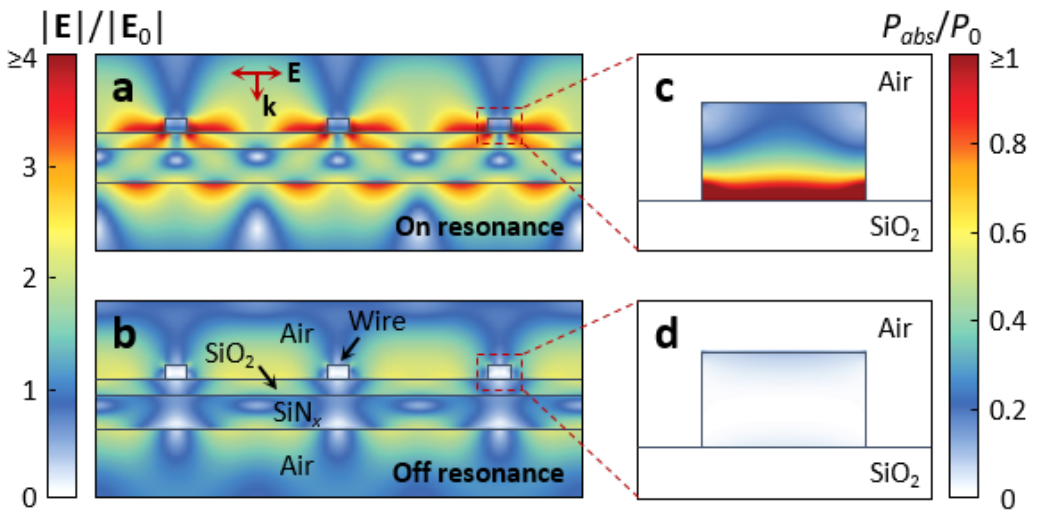

Figure 2.27: Full wave simulations for guided mode resonance structure with dimensions of 40 nm high, 68 nm wide, pitch of 488 nm wires on a waveguide of 50 nm SiO_2/100 nm SiN_x, with Sb_2Te_3 wires (see Figure 2.4 for dielectric function). **a**, At peak absorption, and **b**, at the minimum absorption. **a, b**, Electric field distributions normalized to incident electric field. **c, d**, Power absorption density is calculated by $P_{abs} = \frac{1}{2}\omega\epsilon''|\mathbf{E}|^2$, and is normalized by P_0, the incident power divided by the wire volume.

tunable narrowband absorption over a wide wavelength range by variation of wire array geometrical parameters. We can tune the absorption resonance over the entire visible spectrum at constant waveguide thickness (50 nm SiO_2, 100 nm SiN_x) by varying the wire array pitch (Figure 2.28). Figure 2.29 shows experimental absorption (black dotted, procedure described further in the next chapter), simulated absorption (blue), and simulated best-fit (red). The peak positions in our experiment closely match those predicted by simulations. The best-fit simulation (red) was achieved by fitting the experimental data with altered wire dimensions in simulations (fitting parameters in Table 2.2). Fitting experimental and simulation spectra to a Fano shape (Gallinet and O. J. Martin, 2011) for one wire pitch (Figure 2.25c, Table 2.1), we found that the experimental spectrum exhibited larger damping caused by losses in the wires, which altered the absorption spectrum shape. Pitch_s and Width_s of our fabricated structures were found via SEM imaging and used in simulations for absorption spectra (red line in Figure 2.29). θ_s was a best fit incident illumination angle from simulation. Pitch_f, Width_f, and f were dimensions and incident illumination angle used in a simulation to best fit the experimental data. The best-fit simulation was multiplied by a scaling factor to further fit.

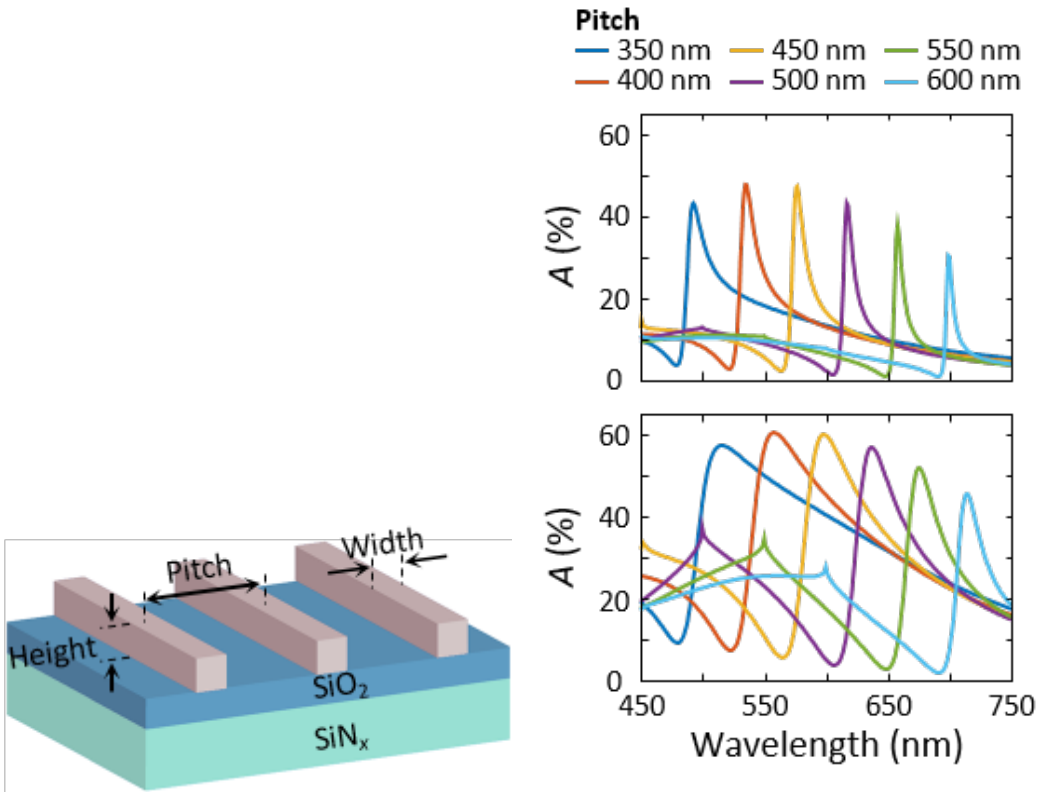

Figure 2.28: (Left) Guided mode resonance structure geometry. Calculated absorption of (Top right) 60 nm wide, and (Bottom right) 100 nm wide wires with thicknesses of 40 nm and varying pitch on suspended 50 nm SiO_2/100 nm SiN_x waveguides.

The absorption maximum can be tuned across several hundred nanometers of wavelength for a given waveguide thickness. Figure 2.31 shows wavelength versus wire pitch for three different SiO_2/SiN_x waveguide thicknesses which display pitch-tunable, narrowband absorption maximum in three different wavelength regimes. Using thicker waveguide layers, Figure 2.31 (center, bottom) show absorption peaks beyond the detection limit of Si photodetectors, which is around 1.1 µm. In principle, the only limitation in infrared tunability for these detectors is the phonon absorption band in SiO_2 (and SiN_x) at around 8-11 μm (Cataldo et al., 2012; Palik, 1997b).

Guided mode resonance incorporated into a thermocouple device

Figure 2.23 show a schematic of our experimental structure, a guided mode resonance wire array, with wire dimensions of 40 nm × 100 nm × 50 µm, in which TM

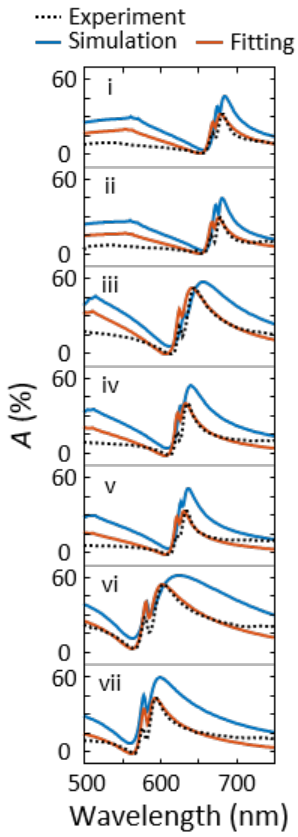

Figure 2.29: Experimental absorption (black dotted), simulated absorption corresponding to the experimental dimensions (blue), and simulated absorption corresponding to fitted and scaled absorption spectra (red) for varying wire pitches and widths on a 45 nm SiO_2/100 nm SiN_x waveguide (see Table 2.2 for dimensions and parameters). Off-normal angle of illumination causes the smaller peak to the left of the larger absorption peak to form (see Figure 2.30).

polarized, normal incident, unfocused optical radiation is coupled into a waveguide mode via a periodic thermoelectric wire array that serves as a light absorber with spectra of the shape shown in Figure 2.32 (blue). Optical power is generated at the thermoelectric junction from absorption in the wires, while the ends of the thermoelectric wires terminate in a broad pad of the same thermoelectric material which reflects most incident light and remains cooler. The resulting temperature difference between the center and edge of the structure is shown Figure 2.32 (orange). Figure 2.33 shows a full wave simulation illustrating the difference in absorption between the pads and wires under unfocused, spatially-uniform illumination. Figure 2.34 shows the difference in power absorbed along a line cut through the length of the simulation in Figure 2.33, which leads to a temperature gradient and results in a

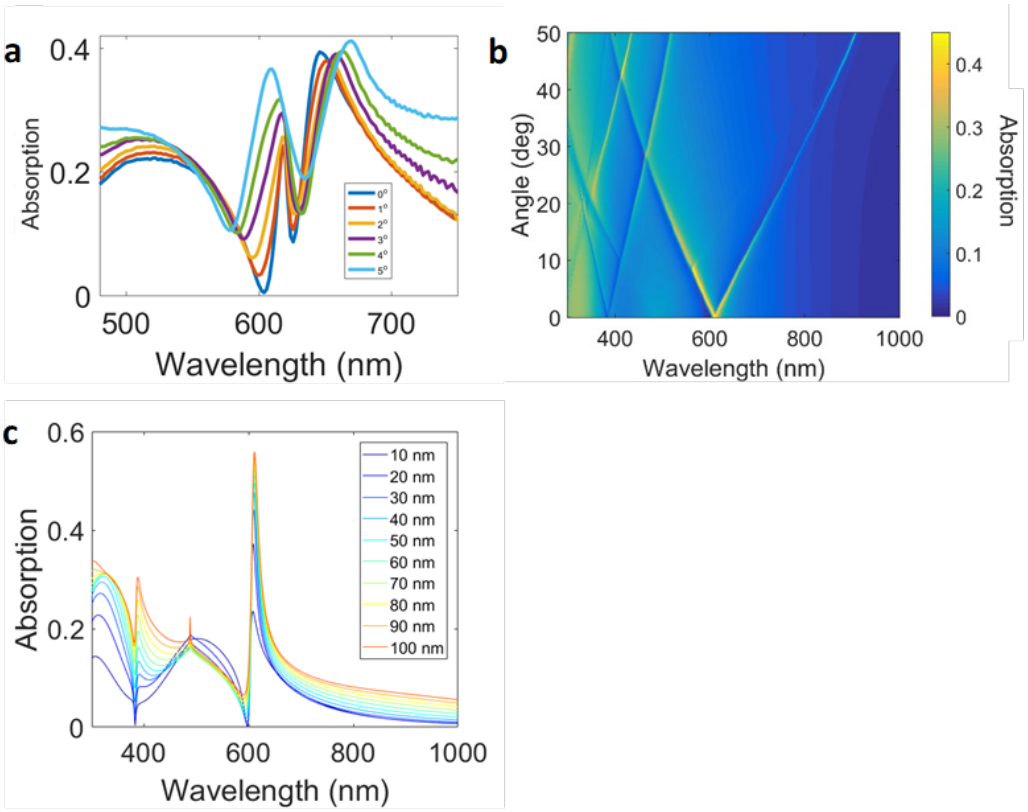

Figure 2.30: Dependence of absorption spectra on incident illumination angle and wire thickness. **a**, Measured absorption spectra for different angles of incidence. An objective with numerical aperture 0.14 was used, giving an angular spread even at normal incidence, producing the side peak at normal (0°) incident angle. **b**, Full wave simulations of the incident illumination angle dependence of 40 nm tall, 67 nm wide Sb_2Te_3 wires with a pitch of 488 nm on a 50 nm SiO_2 on 100 nm SiN_x waveguide. Even at 1 degree offset, the single peak splits into two. **c**, The dependence of wire thickness on absorption spectra, with pitch of 488 nm. The absorption asymptotes to its maximum value for wire heights around 40 nm.

thermoelectric voltage (TEV). Our nanophotonic thermoelectric structures on thermally insulating membrane substrates have dimensions large enough that bulk heat transport equations can be used (i.e. no ballistic or quantized thermal conductance). To maximize responsivity, we seek to maximize the TEV, which is proportional to the Seebeck coefficient, , and the temperature difference, ΔT, between cold and hot ends of the material, i.e., $TEV = \alpha \Delta T$. α is primarily dependent on material and deposition methods, while nanostructuring has been shown to alter the Seebeck coefficient to some degree (W. Wang and Z. M. Wang, 2014; Koumoto and Mori, 2013). ΔT can be increased through five primary design approaches. First, high

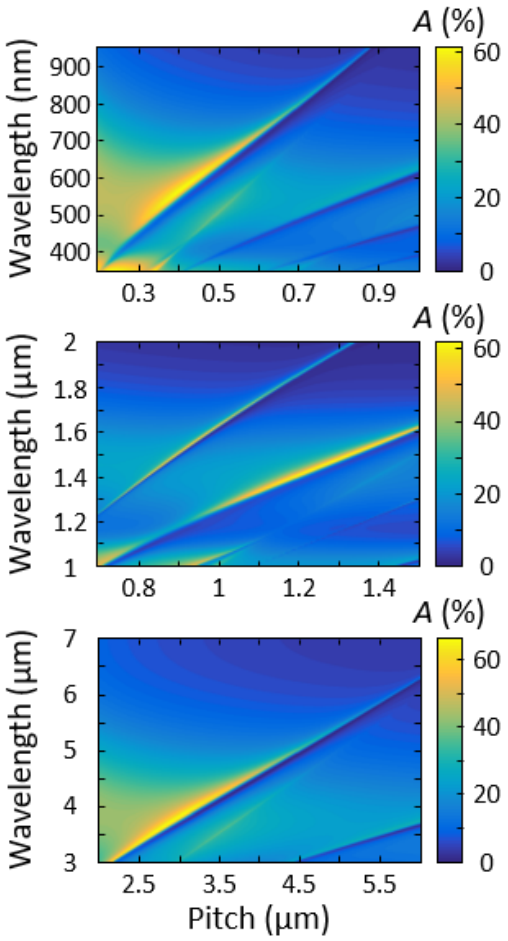

Figure 2.31: (Top) Wavelength versus pitch absorption plot in the visible regime for 40 nm thick Sb_2Te_3 wires, on a 50 nm SiO_2/100 nm SiN_x suspended membrane. (Center) Absorption spectra for 50 nm thick, 300 nm wide Sb_2Te_3 wires on a 300 nm SiO_2/500 nm SiN_x suspended membrane. (Bottom) Absorption spectra in the mid-IR for 50 nm thick, 1.5 µm wide Bi_2Te_3 wires on a 500 nm SiO_2/500 nm SiN_x suspended membrane. All calculations use either Sb_2Te_3 or Bi_2Te_3 as the wire material (see Figures 2.2, 2.3,2.4, 2.5 for dielectric functions). Simulations were performed first coarsely with Lumerical FDTD (Lumerical, n.d.), then were refined by Seyoon Kim with finer pitch steps using an RCWA method based on (Moharam et al., 1995).

light absorption in the desired "hot region" is essential. Secondly, low energy loss via radiation (i.e. low emissivity) in the "hot region" is desirable, with higher emissivity in the "cold region". Thirdly, low conduction through the interface is preferred, via suspending the thermoelectric "hot region" or having high thermal interface resistance. Fourth, as with any thermoelectric device, a low thermal conductivity is necessary to maintain a high temperature gradient, achieved by material

selection, nanostructuring, or by choice of deposition methods. Finally, low convective losses to the surrounding gas ambient in the "hot region" are preferred, and can be achieved by operation of the thermoelectric structure in vacuum (although the loss of convection in the "cold region" could be detrimental to a temperature gradient and should be carefully considered).

As an example of a thermoelectric plasmonic nanostructure, we consider a periodic array of wires composed of thermoelectric materials on a thin, suspended, electrically insulating, low thermal conductivity substrate. Using the electromagnetic power absorption simulations as inputs, we can simulate the temperature profiles in our structures; an example is shown in Figure 2.35. Temperature difference ΔT as a function of wire length is shown in Supplementary Fig. 1d. Longer wires produce a larger temperature difference for a given power density, but will have a larger resistance, increasing the Johnson noise and therefore increasing noise equivalent power (NEP), shown in Figure 2.37f. Additionally, smaller structure sizes are preferable, e.g., for camera pixel applications, motivating us to choose a wire length of 50 μm, which shows reasonable responsivity for the chosen power densities and yields a low NEP. A wire array/substrate heterostructure supporting guided mode resonances in an n/p-type TE junction is shown in Figure 2.36. The absorption resonance can be spectrally shifted by several hundred nm by varying the wire array period. Thus, a periodic tiling of wire array pixels each with a different period and resonance frequency could function as a thermoelectric hyperspectral detector, shown conceptually in Figure 2.38.

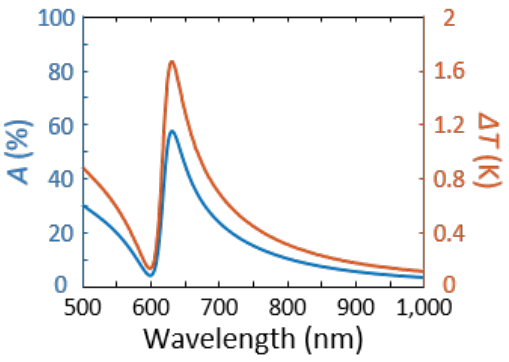

Figure 2.32: Theoretical absorption (blue) and temperature difference between the center of the wire and edge of the pad (orange) for a structure with 40 nm tall by 100 nm wide Sb_2Te_3 wires spaced 488 nm apart.

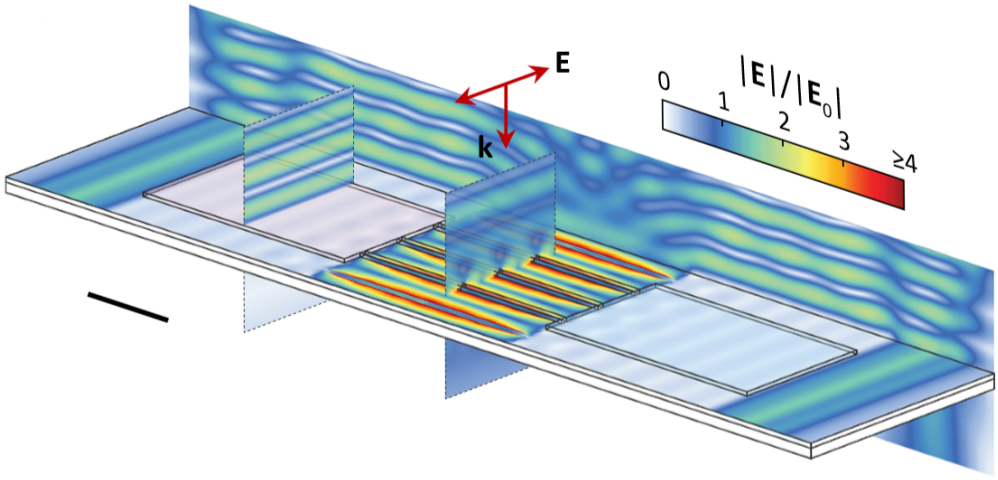

Figure 2.33: Electric field profile normalized to incident electric field of a periodic structure at peak absorption (1 μm scale bar). Highest |**E**| occurs in the wires, leading to absorption, while the pads largely reflect light creating the necessary temperature gradient.

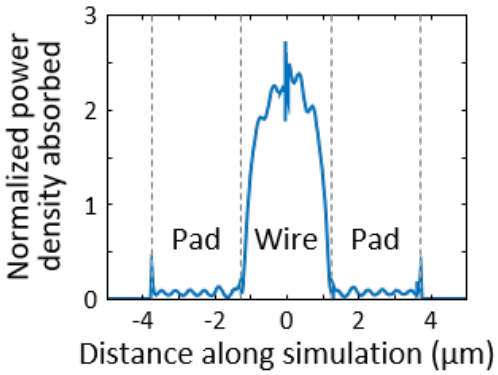

Figure 2.34: Power density absorbed along a line cut through the simulation in Figure 2.33. Asymmetry arises from half of the device being Sb_2Te_3 and the other half being Bi_2Te_3. Power density is normalized to incident power divided by thermoelectric structure volume.

Thermal design

The theoretical temperature spectrum found in Figure 2.32 for the guided mode resonance wire structure was constructed as follows. Absorption data were taken from 2D periodic electromagnetic simulations and combined with experimentally measured power. The combined data was used as a volumetric input power source in a thermal simulation. Simulation sizes were large enough that changes in fixed boundary temperatures did not affect central temperature profiles. Due to memory

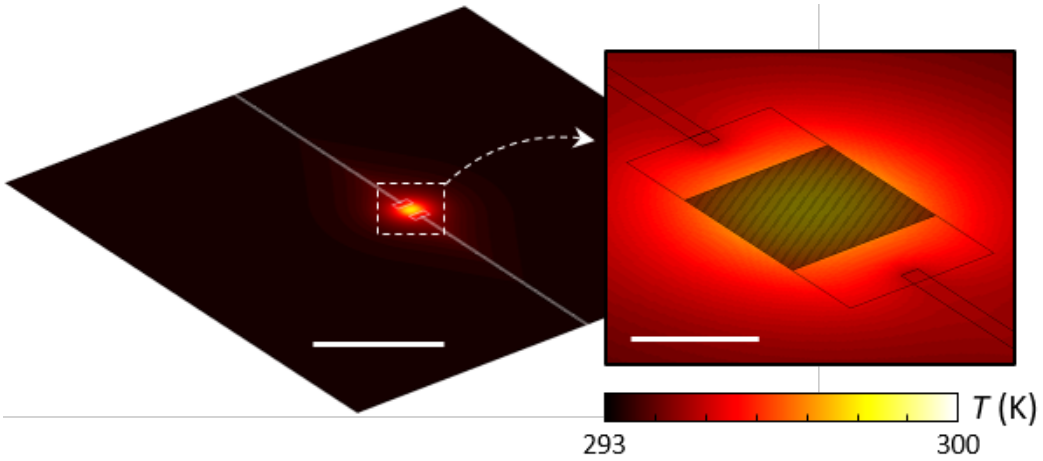

Figure 2.35: A thermal simulation of the Bi_2Te_3 - Sb_2Te_3 structure at peak absorption with input power of 20 μW. Scale bar is 500 μm, inset scale bar is 50 μm.

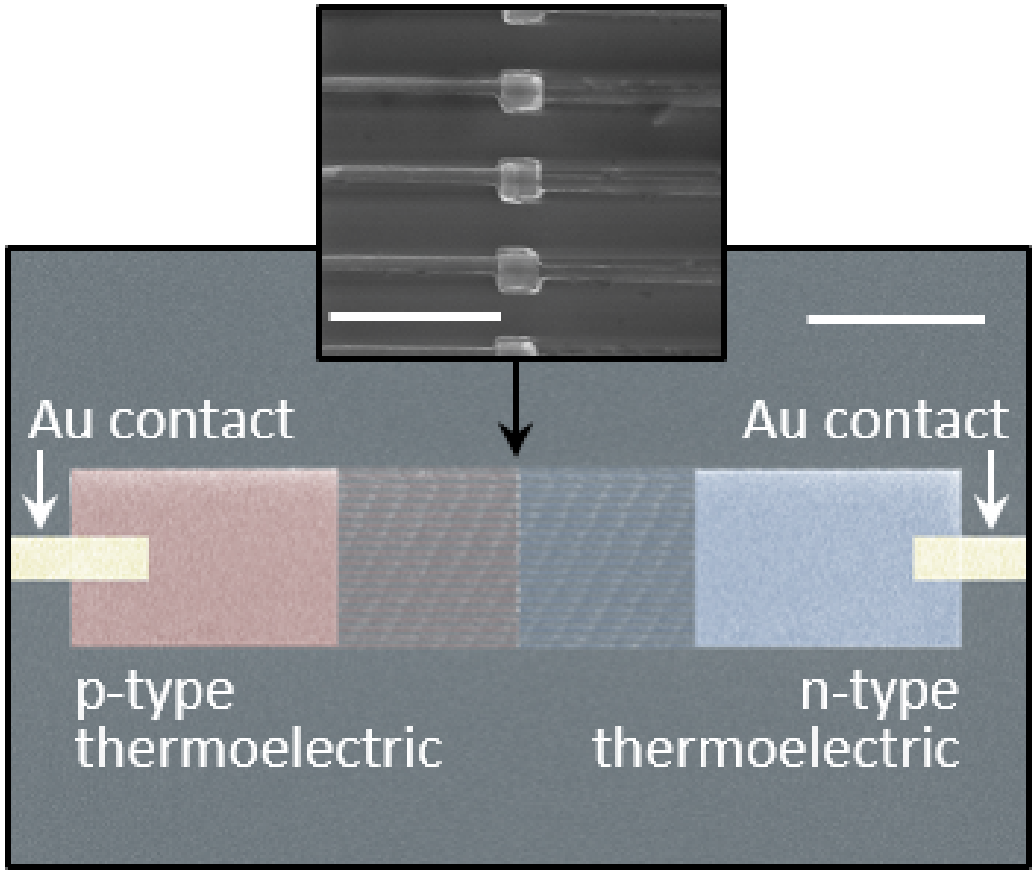

Figure 2.36: False color SEM of a fabricated p/n thermoelectric structure, with Au contacts (20 μm scale bar). Inset is the junction between Bi_2Te_3 - Sb_2Te_3 wires (1 μm scale bar).

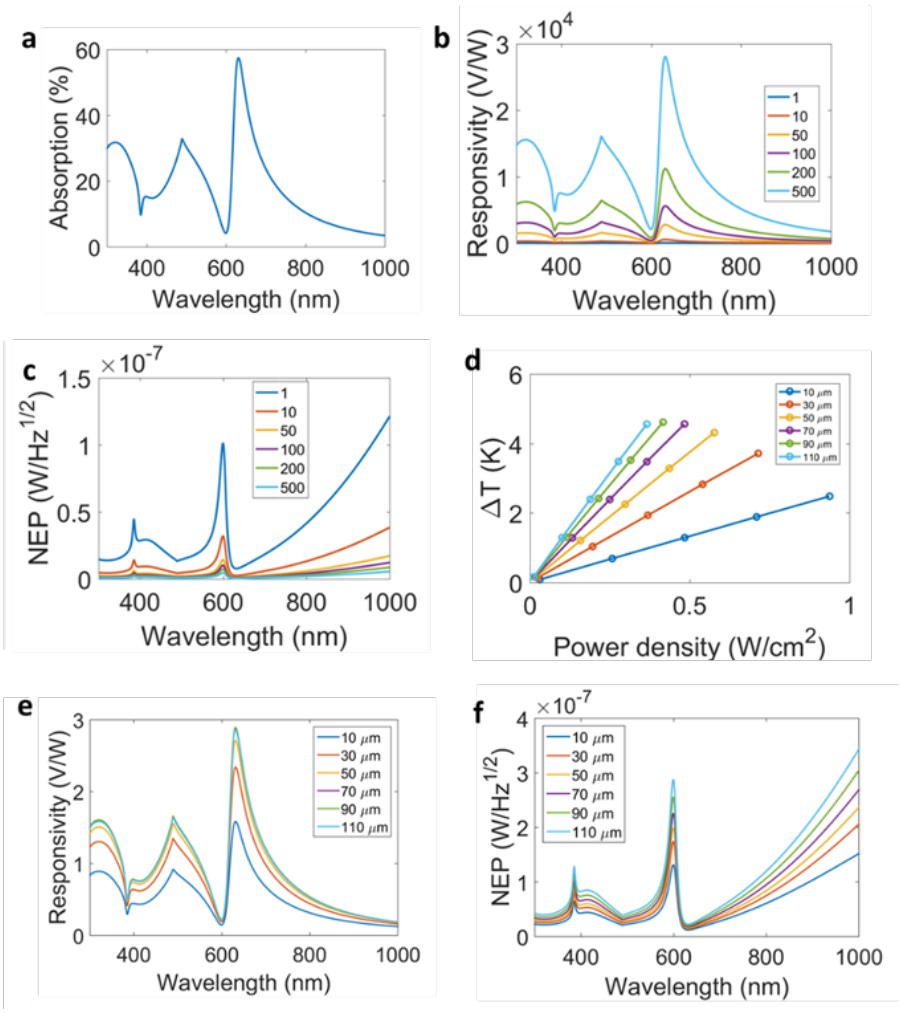

Figure 2.37: Thermopile and wire length analysis. **a**, Absorption spectra used in simulations for **b,c,e,f**. **b**, Responsivity for a structure with 50 μm long wires and 27 μm long pads, as a function of number of wires in a thermopile configuration. The entire structure is illuminated and responsivity is calculated relative to power striking the wire area. The pads are assumed to have a 20% absorption, independent of wavelength. **c**, Noise equivalent power (NEP) for the thermopiles in **b**, assuming Johnson noise as the noise spectral density using simulated average temperatures. **d**, Temperature difference between the edge of the pad and the center of the wires versus power density for different wire lengths. Pad sizes remain constant with dimensions of 50 μm by 27 μm by 50 nm. **e**, Responsivity as a function of wavelength for the absorption spectrum in **a**, for different wire lengths, relative to power illuminating the entire structure. **f**, Noise equivalent power corresponding to the responsivity in **e**, for different wire lengths. Noise spectral density is theoretical Johnson noise using simulated average temperatures for the structures. Simulation details are given in the text.

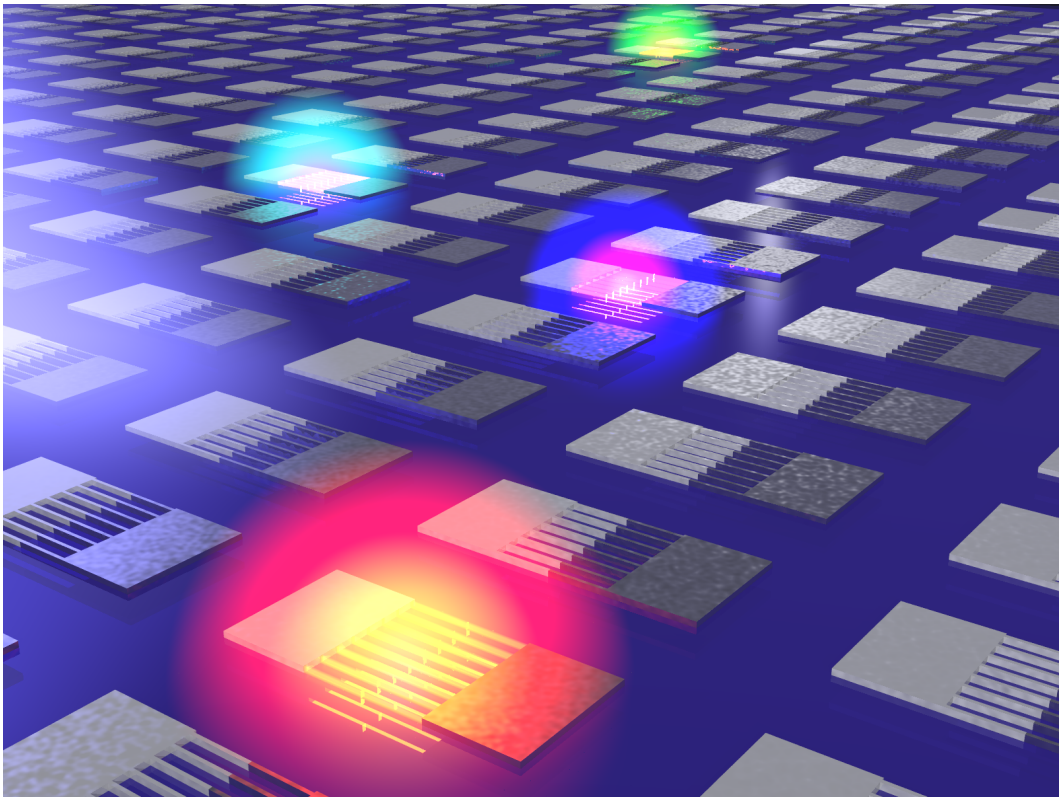

Figure 2.38: Conceptual design of hyperspectral pixel. Each thermoelectric structure in the pixel has a different wire pitch, causing each structure to have an absorption peak that corresponds to a different wavelength (Figure 2.31). When light of unknown wavelength content illuminates the pixel, voltages will be produced in each structure depending on their specific absorption profiles. Deconvolution of these voltage signals through an algorithm can enable identification of the unknown wavelengths.

constraints, symmetry boundary conditions were used, and so simulations used one material, bismuth telluride, instead of both bismuth telluride and antimony telluride.

Of course, while our thermal simulations were able to explain experimental results reasonably well (compare Figure 3.4 and Figure 3.12j-l), it is known that small crystal grain sizes can decrease thermal conductivity in thin film BiTe-based materials (Takashiri et al., 2008). From our XRD data (Figure 3.1), we can see that the grain structures must not be large. We recognize the bulk values of thermal conductivity used in our thermoelectric materials in the simulations may be different than that of our thin-film materials. Dimensions in simulations were the same as those of the fabricated sample: 40 nm tall wires and pads, 100 wires total, 50 μm long (25 μm for each thermoelectric material), with a pitch of 520 nm and wire width of 130 nm.

The thermoelectric pad lengths were 27 μm and as wide as the wire array. The Au contacts were 6 μm wide and 70 nm tall, and overlapped the pads by 5 μm.

2.9 Guided mode resonator thermopile

Nanophotonic design

Thermopiling would further increase the responsivity of the guided mode resonance design. Figure 2.39 shows an artist's rendition of this idea. The nanophotonic response would be nearly identical to the thermocouple case, as the pads would still be wide enough to act like bulk materials and reflect light (Figure 2.39). The absorption for the thermopiled structure wires is shown in Figure 2.37a.

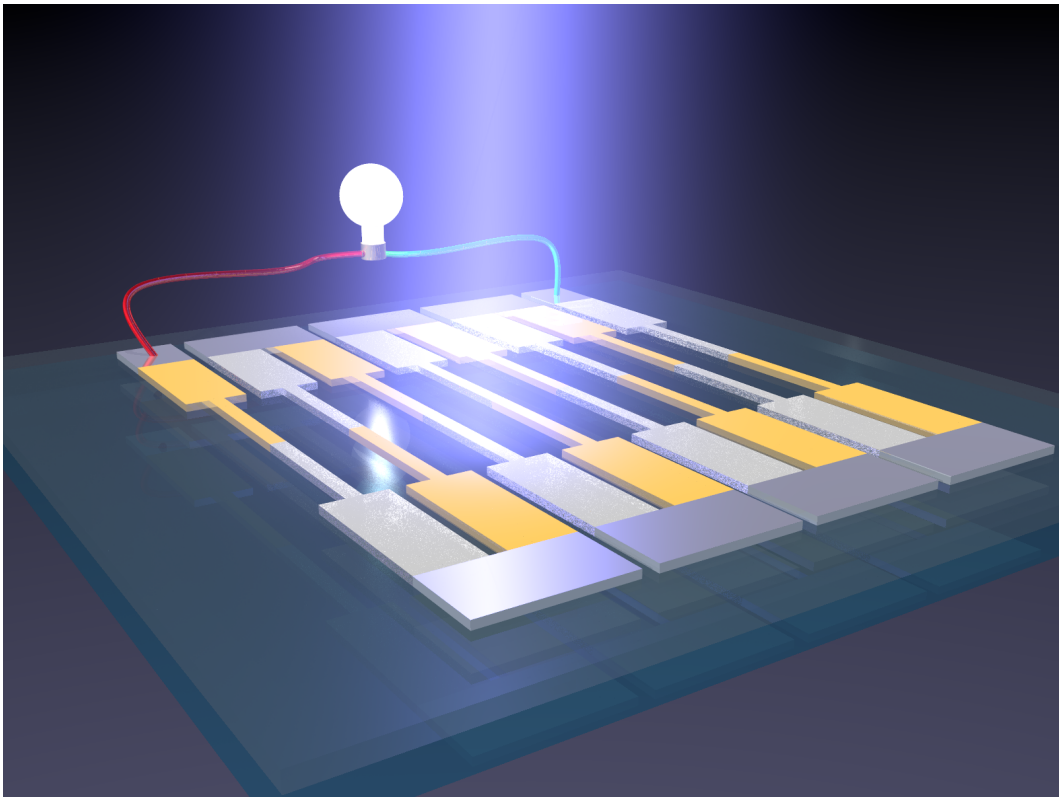

Figure 2.39: Artistic rendition of a thermopiled guided mode resonance structure, when p-type and n-type thermoelectrics are alternated in series. Image rendered in POV-ray.

Thermal design

Simulations done in Figure 2.37 were performed in a similar manner to those described for the guided mode resonance wire structures above. The results of the wire simulations were also used in Figures 2.37b,c to approximated outputs of thermopile

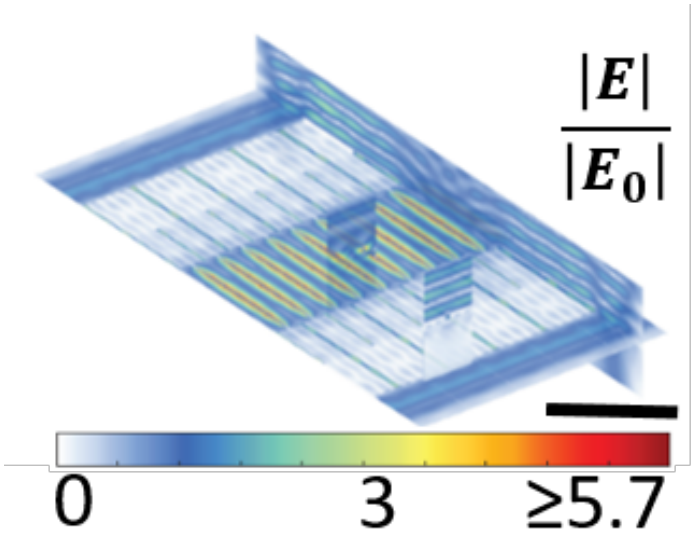

Figure 2.40: Full wave simulations of the thermopiled guided mode resonance structure, with electric field |**E**| normalized by incident electric filed |$\mathbf{E}_0$|. Excitation wavelength is 631 nm (peak absorption for this geometry).

configurations. We find that increasing power density will linearly increase the difference in temperature between the "cold" edge of the thermoelectric pads and the junction at the center of the wires (Figure 2.37d). Relative to power impingent upon the entire structure, the responsivity of wires increases with length until 90 µm, then slightly decreases at 110 µm (Figure 2.37e, average of measured Seebeck coefficients of –84 µV/K (Bi_2Te_3) and 242 µV/K (Sb_2Te_3) used). We can explain the decrease in responsivity at 110 µm in the following way: while there is more area available to absorb light, and therefore more total power absorbed, this absorbed power is not localized on the thermoelectric junctions as much as the absorbed power in the smaller wire structures is. Because pad lengths are held constant, the larger wire area will simply heat up the entire structure, including the edges of the pads more. The maximum heat of the wires will increase, but the temperature of the "cold" ends will also increase, creating a smaller overall ΔT, and therefore a smaller voltage. Increasing the length of the pads would allow responsivity to increase with wire length for wires longer than 110 µm. Using theoretical Johnson noise as our noise spectral density and using the measured resistivity of our materials and simulated temperature rises, we find that because of their overall higher resistances, longer wires will have higher noise equivalent powers. We chose to study structures with 50 µm wires as they give a high responsivity with lower noise equivalent power.

2.10 Analysis and summary of simulation findings and guidance for future designs

Noise considerations

The perfect absorber and split ring resonator designs studied in this chapter theoretically could have higher detectivities than the guided mode resonance wire structures described and tested in the later part of the chapter and the next chapter. For the perfect absorber (Figure 2.1) and split ring resonator absorbers (Figure 2.11), we have designed a thermal management scheme to provide motivation for future research. The nanophotonic absorber arrays were placed in the center of a thin thermoelectric junction on top of a thin membrane of SiN_x, described further in Figures 2.7. The simulation was done without any assumed thermal convection in the surrounding ambient. The supporting edges of the thermoelectric 'bridge' structure depicted in Figure 2.7 were placed on a laminate of SiN_x on a thin Au film on a thick layer of SiO_2. The high thermal conductivity of the thin Au layer greatly lowered the "cold side" temperature of the thermoelectric material to near room temperature, compared to when this layer was not present. This allowed us to increase the thermal gradient within the thermoelectric materials, creating higher responsivities than in our resonant wire structures, from 180 to 390 V/W (Figures 2.9, 2.15). Using the measured resistivities for our structures, ($2.75\times10^{-2} \pm 7.2\times10^{-5}$ Ω·cm for Bi_2Te_3 and $4.27\times10^{-1} \pm 1.1\times10^{-3}$ Ω·cm for Sb_2Te_3, discussed later) and the dimensions described in Figure 2.7, the resistance of the thermoelectric element would be around 91 kΩ in the perfect absorber, and 152 k in the split ring resonator, giving a theoretical Johnson noise of 38 nV/Hz1/2 and 50 nV/$Hz^{1/2}$, respectively. Thus, the minimum noise equivalent power (NEP) would be 0.2 nW/$Hz^{1/2}$ for the perfect absorber and 0.13 nW/$Hz^{1/2}$ for the split ring resonator absorber, shown in Figures 2.10, 2.16. This is 10 to 20 times less than our measured and simulated NEP minima of our structures, which is significant.

Resonant thermoelectric nanophotonics versus conventional thermoelectric detectors

The distinction between a conventional (c.f. (F. Völklein and A. Wiegand, 1990)) thermopile and a resonant thermopile (Figure 2.39) lies in the total heat capacity of the hot end of the structure. A smaller heat capacity (i.e. smaller structure) will generally have both higher responsivity and smaller time constant for the same amount of incident power absorbed. Resonant absorbing structures can absorb a large fraction of light in small volumes because their absorption cross-section is much larger

than their geometric cross-section, unlike conventional absorbing layers such as black paint. Furthermore, layers like black paint, which rely on single-pass material absorption, need to be thicker than the skin depth of the incident light, whereas this is not a requirement for resonant structures which rely on multiple-passes of light. If the thermopile detector takes advantage of absorption from multiple passes of light with non-thermoelectric absorbers (A. S. Gawarikar, R. P. Shea, and J. J. Talghader, 2013a), the heat capacity for this structure will still be higher than if the thermoelectric materials themselves were the absorbers, and still gives a 10's of milliseconds time constant, as there is simply more material if an absorber plus a thermoelectric element is needed. Thus, if equal amounts of light are absorbed, thermoelectric structures which themselves absorb (especially suspended, isolated thermoelectric absorbers) have the potential for higher responsivity with lower time constant than devices with non-thermoelectric absorbing structures, because of the smaller volume and therefore heat capacity of the materials involved.

Take as an example the thermopile structure in (F. Völklein and A. Wiegand, 1990). This structure relies on an absorbing layer on the order of microns thick and 500 μm in diameter inside a circle of thermopiles. If we have the guided mode resonant thermoelectric thermopile structure shown in Figure 2.39 that uses thermoelectric material as the absorber, and assume the thermoelectric materials absorb an equivalent amount as the membrane, then the maximum temperature will occur in the thermoelectric wires themselves, but with much less material than in the membrane in (F. Völklein and A. Wiegand, 1990). This will lead to a higher responsivity and lower time constant in the resonant thermoelectric structure, although the noise spectral density may be higher due to larger resistance in the structure. Our present design has an absorption maximum of about 60% in the thermoelectric wires, but by altering the design slightly to form a Salisbury screen (Fante and McCormack, 1988), 100% absorption can be reached in the thermoelectric materials.

Chapter 3

HYPERSPECTRAL DETECTOR APPLICATION

We propose and demonstrate here nanostructures composed of thermoelectric thermocouple junctions using established thermoelectric materials – Bi_2Te_3/Sb_2Te_3 – but patterned so as to support guided mode resonances with spectrally sharp absorption profiles. Spatially localized absorption in resonant thermoelectric nanophotonic structures results in localized heating of the thermoelectric material, generating large thermal gradients under unfocused optical excitation. We find that the small heat capacity of optically resonant thermoelectric nanowires enables a fast, 337 μs temporal response, 10-100 times faster than conventional thermoelectric detectors. We show that TE nanophotonic structures are tunable from the visible to the mid-IR, with small structure sizes of 50 μm by 110 μm. Whereas photoconductive and photovoltaic detectors are typically broadband (with exceptions noted, e.g. (Mokkapati et al., 2015; Chang et al., 2007; Wipiejewski, Panzlaff, and Ebeling, 1992)) and are insensitive to sub-bandgap radiation, nanophotonic thermoelectrics can be designed to be sensitive to any specific wavelength dictated by nanoscale geometry, without bandgap wavelength cutoff limitations or need for cooling. From the point of view of imaging and spectroscopy, they enable integration of filter and photodetector functions into a single structure.

3.1 Background of hyperspectral detectors

Hyperspectral detectors detect light with much finer wavelength resolution than RGB detectors. Originally developed by NASA and JPL (Goetz, 2009) for space-based geology analysis and military applications, hyperspectral detectors have found uses from art conservation (Fischer and Kakoulli, 2006), to wetland vegetation identification (Adam, Mutanga, and Rugege, 2010), to food safety (Gowen et al., 2007). Additionally, hyperspectral imaging has found applications in medical imaging (Lu and Fei, 2014), as different tissues (such as healthy and cancerous) absorb and emit different wavelengths of light which RGB imaging may not have fine enough wavelength resolution to elucidate.

Hyperspectral imaging is more complicated than RGB imaging. First, some form of narrow-band filter must be used to descriminate between different wavelengths of light. This decreases the incident power of light, so detectors must be have a

good signal to noise ratio. Photovoltaic-based materials are also band gap limited, so a single material, like Si, has a wavelength cutoff of 1100 nm. Thus, a visible to IR hyperspectral detector would need several different materials to detect the entire spectrum, and the small bandgap materials would need cooling to maximize sensitivity.

3.2 Fabrication and materials analysis

Fabrication

The thermoelectric hyperspectral detectors were fabricated as follows. On top of the waveguide layer of 100 nm thick SiN_x membrane (Norcada NX7150C), the 45 or 50 nm SiO_2 spacer layer was deposited via PECVD (Oxford Instruments System 100 PECVD) at 350°C.

The structures were written via electron beam lithography (Raith EBPG 5000+) in a series of aligned writes, followed by deposition and liftoff. In order to spin resist on the membrane, a sort of "stilts" were constructed to allow airflow under the membrane, using a method designed by Seyoon Kim. If the membrane were directly glued to a substrate, when put under vacuum, the air bubble trapped below the membrane will burst the membrane if not allowed to escape. These "stilts" were composed of two thin silicon chips that were first glued to a larger silicon wafer, and the membrane was placed across these thin silicon chips so that air could flow below the membrane. The "stilts" were attached the the larger silicon wafer by spinning PMMA 495 A8 at 1100 RPM for one minute, placing the "stilts" on the wafer, and baking at 180°C for 5 minutes. Then PMMA 495 A8 was spun on top of the "stilts", again at 1100 RPM for 1 minute. The membrane was then placed on the "stilts", and baked at 180°C for 5 minutes to adhere. Next, a bilayer resist was spun on the membrane to aid in liftoff. PMMA 495 A4 was spun at 4,000 RPM for one minute, then baked at 180°C for 5 minutes, followed by PMMA 950 A2 spun at 4,000 RPM for one minute, then baked at 180°C for 5 minutes. The idea behind a bilayer resist is that the lower resist, the PMMA 495 A4, will develop move than the top layer under an electron beam, creating an overhang. This overhang will prevent deposited material from adhering to the sidewalls of the resist, which prevents liftoff. This process works very well for directional deposition, such as electon beam evaporation or thermal evaporation with a long throw distance (i.e. material impinges primarily normal to the surface). This directionality is due first to, geometry, due to the long distance from source to sample, and second to low base pressure in the chamber (10^{-5} - 10^{-9} Torr generally), which means evaporated

material has zero or few collisions between evaporation and landing on the sample due to the long mean free path of atoms at these low pressures. Sputtering, on the other hand, is very non-directional. Sputtering is very effective at evenly coating surfaces for several reasons. First, the throw distance in sputterers is generally must be much shorter than in electron beam or thermal evaporators. This is due to the higher pressure during deposition needed to sustain a plasma (around 10^{-3} - 10^{-2} Torr), so the mean free path of particles is much shorter, and cannot travel for before being redirected by a collision with another particle or gas particle. These frequent collisions additionally allow materials to slowly coat the inside of resist overhangs, making liftoff more difficult, although not impossible.

Alignment markers were 20 by 20 micron squares composed of an electron beam evaporation (Kurt J Lesker Labline or CHA Industries Mark 40) deposited 5 nm Ti adhesion layer (0.5 Å/s) and 55-70 nm of Au (1-1.5 Å/s), and lifted off in Acetone. 40 nm of Bi_2Te_3 and Sb_2Te_3 were magnetron sputter deposited (Kurt J Lesker) with 40 W RF power (US Guns). 40 nm of alumel and chromel were magnetron sputter deposited with 500 W DC power (AJA Orion). Contacts were deposited in the same way as alignment markers, but with closer to 100 nm of Au. In this experiment, Ti was used as an adhesion layer to the thermoelectric materials (Ni was used as an adhesion layer for cloverleaf Hall measurement samples for Bi_2Te_3).

The liftoff process could be done fast or slow. The fast process involved submerging the sample in acetone on a 70°C hot plate. After approximately 5-15 minutes, depending on the deposition method used (the Lesker Labline took 5 minutes, the CHA took closer to 15 minutes), the edges of the chip were scraped to aid the liftoff process. This was done when the sample surface started to look "wrinkly". After about 5 more minutes, acetone was forcibly sprayed on the sample (avoiding directly sqirting the membrane) to help loosen the remaining material. The material was generally lifted off after about 30 minutes. Occasionally, letting the sample soak for several days would aid in lifting off more material, but this usually was not the case. The slow liftoff process involved allowing the sample to sit covered in room temperature acetone overnight. Liftoff success was extremely variable, especially with cosputtered and thermally evaporated thermoelectric materials, explored in later chapters. Bismuth-heavy materials lifted off much worse than tellurium-heavy materials. This could be due to surface tension differences.

Fabricated dimensions of Figure 2.36 are 100 nm wide, 40 nm thick, wires with a 470 nm period, fabricated on a 145 nm thick freestanding dielectric slab waveguide

composed of 45 nm SiO_2 and 100 nm SiN_x layers.

Bismuth telluride and antimony telluride compositional and structural analysis and materials characterization

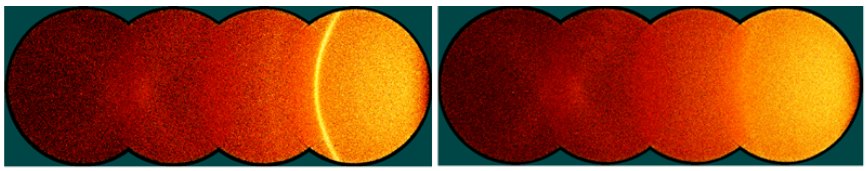

Figure 3.1: XRD data of 100 nm of Bi_2Te_3 (left) and 50 nm Sb_2Te_3 (right) show very little crystallinity, as sputtered in experiments. Two-dimensional diffraction image frames were collected with frame centers set to 20, 40, 60 and 80 degrees in 2Θ, from right to left, and then merged. Credit to Slobodan Mitrovic for data and analysis.

Figure 3.1 shows XRD data on thin films of Bi_2Te_3 and Sb_2Te_3, deposited from a stoichiometric target as described in the previous section. We notice that in both cases the data show signatures of nanocrystallinity or perhaps even amorphous structure in the case of antimony telluride.

Two-dimensional XRD data were collected with a Bruker Discover D8 system, with a Vantec 500 detector and Cu Kα x-ray line produced by a microfocused IμS source, in a $\theta - 2\theta$ measurement. Figure 3.1 shows results on thin films of Bi_2Te_3 and Sb_2Te_3.

X-ray photoemission spectroscopy (XPS) was performed on a Kratos Nova (Kratos Analytical Instruments), with monochromatized x-rays at 1486.6 eV and using a delay-line detector at a take-off angle of 35 degrees. The pressure during measurement was better than $5x10^{-9}$ Torr, and the data were collected at 15 mA and 15 kV from an area of about 0.32 mm^2. Survey scans were collected at pass energy 160, and high-resolution scans at pass energy 10. Figure 3.2 shows XPS survey spectra for our Bi_2Te_3 and Sb_2Te_3 thin films, and due to surface sensitivity of the technique, these represent only the top few nanometers of the sample.

Apart from expected surface oxidation and hydrocarbons from air, there are no other contaminants present that could affect the Seebeck coefficient. The stoichiometry of bismuth telluride and antimony telluride greatly affects the Seebeck coefficient(Horne, 1959; Bottner et al., 2004; Silva, Kaviany, and Uher, 2005). Bismuth

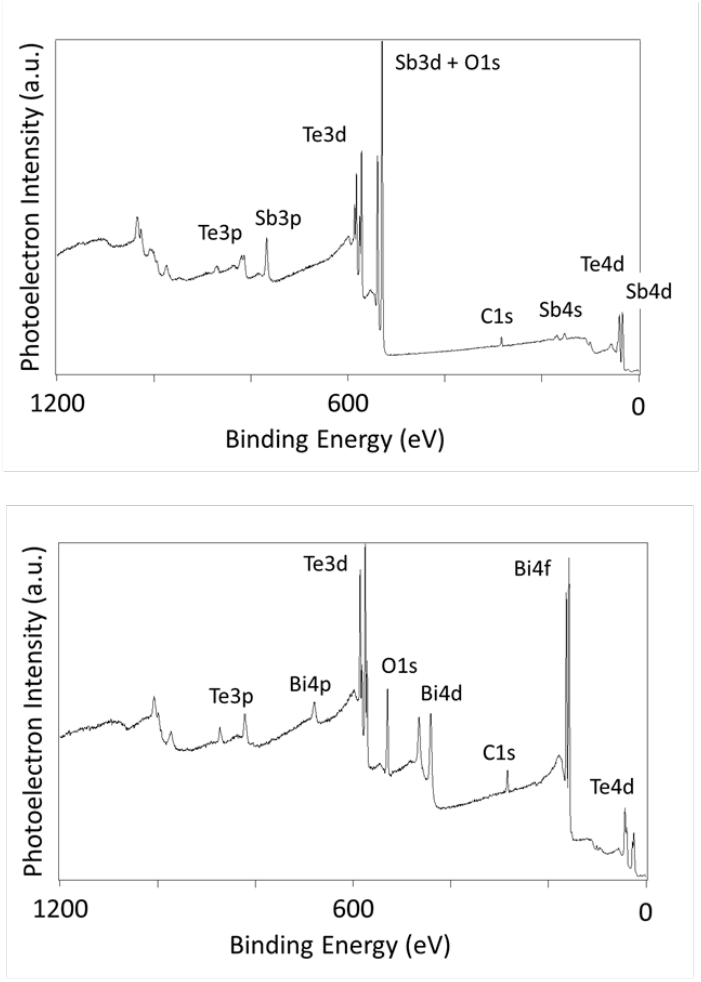

Figure 3.2: XPS survey scans. Sb_2Te_3 (top) and Bi_2Te_3 (bottom) samples. Credit to Slobodan Mitrovic for data and analysis.

telluride is particularly sensitive: a small variation in atomic percent composition is capable of changing the carrier type from electrons to holes(Horne, 1959).

Using quantitative analysis based on Te 3d and Bi 4f levels, shown in Figure 3.3, we determined that the composition of our bismuth telluride was 42.5%:57.5%, Bi:Te for surface relative concentrations. This corresponds to a wt% of about 53.7% for the bismuth.

The XPS measured composition of our 50 nm antimony telluride film was determined from Sb 3d 3/2 and Te 3d 3/2 peak areas (as identified to belong to the compound), and indicates a composition of 32%:68% Sb:Te. A large amount of antimony on the surface had oxidized.

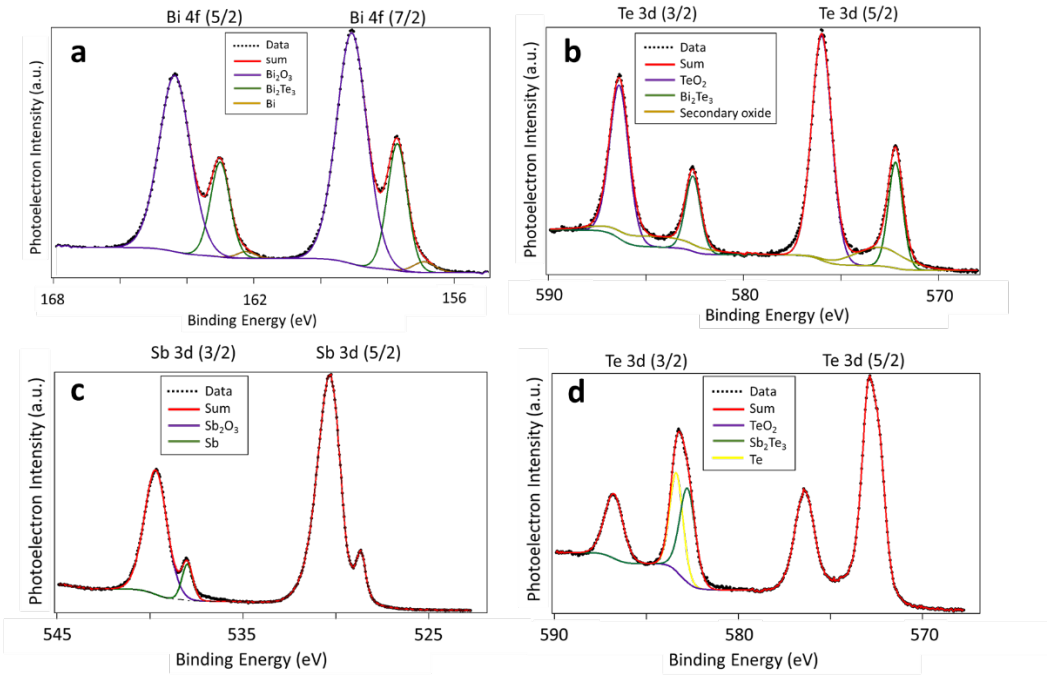

Figure 3.3: Compositional analysis. Detailed XPS data and fits for bismuth telluride peaks (**a-b**) and antimony telluride peaks (**c-d**) for our samples. **a**, Three components are visible in Bi 4f levels: the major components are Bi_2O_3, and Bi_2Te_3 (157.1 eV and spin-orbit pair at + 5.3 eV), with a small amount of elemental bismuth (156.6 eV). **b**, Te 3d level in bismuth-telluride is mostly TeO_2 and Bi_2Te_3 (582.3 eV, SO-splitting of 10.4 eV). **c**, Sb 3d levels show that most of the surface of antimony-telluride is oxidized (Sb_2O_3), much more so than the bismuth-telluride, with a measurable Sb_2Te_3 component (538 eV). **d**, Te 3d levels in antimony-telluride show the telluride, an oxide and elemental Te peaks. Credit to Slobodan Mitrovic for data and analysis.

Seebeck measurements at room temperature were performed based on a thin film Seebeck measurement technique described in the literature (Singh and Shakouri, 2009). The sample thermoelectric materials were deposited on 500 nm layer of SiO_2 on a 1 cm by 1 cm silicon chip. The thermoelectric materials were of the same thickness of the structures used in this paper. Copper blocks were placed with thermally conductive paste on the hot and cold sides of a thermoelectric heating stage with the thin-film sample straddling the copper blocks, attached with thermally conductive paste. Thin K-type thermocouple wire junctions were placed on the surface of the thermoelectric material on the hot and cold ends, using physical pressure. Soldering or using thermally conductive paste to connect the wires to the samples was not recommended (Singh and Shakouri, 2009), and we measured large hysteresis when using thermally conductive paste. A voltmeter (Keithley 6430 sub-

femtoamp remote source meter) was attached to the chromel end termination of each thermocouple before it was passed into a temperature meter, providing temperature and voltage data from the exact same location on the sample. Stage temperature was measured using a build-on thermocouple attached to the stage, which roughly verified the sample temperature measurements were in the correct range. Next, the Seebeck coefficient of the chromel wire used as half of the thermocouple/electrode was measured in a similar manner to subtract out its contribution to the thin film voltage measured. The thin film samples were not measured in vacuum, which could contribute error to the temperature measurements, as convection could slightly cool thermocouples used in the measurements relative to the sample surface. Data from several specimens of each material were collected, and is displayed in Table 3.1. Variability among specimens deposited at the same time in the same chamber can be accounted for by distribution of substrate positions within the deposition system. There was no substrate rotation during the deposition, which leads to small thickness variations or unequal substrate temperatures amongst our samples. Amorphous thin films of Sb_2Te_3 have shown a variation of 100 μV/K when deposited under the same conditions (Baily and Emin, 2006), indicating that our small sample-to-sample variation is not unique.

Sample	Seebeck coefficient (μV/K)
Sb_2Te_3 – 1	264 ±16.6
Sb_2Te_3 – 2	214 ±14.6
Sb_2Te_3 – 3	247 ±15.9
Bi_2Te_3 – 1	-81 ±3.1
Bi_2Te_3 – 2	-77 ±3.9
Bi_2Te_3 – 3	-95 ±5.2

Table 3.1: Measured Seebeck coefficient of 6 different samples of thermoelectric material. To clarify the sign convention, Sb_2Te_3 here is p-type and Bi_2Te_3 is n-type.

Seebeck coefficient is theoretically determined primarily by crystal band structure. For quintuple layers (unit of Te-Bi-Te-Bi-Te or Te-Sb-Te-Sb-Te) of bismuth telluride and antimony telluride, it has been calculated that band structures will be altered for less than 12 quintuple layers, due to opening of a bandgap attributed to topological surface-states, but this gap will close with more than 12 quintuple layers and approaches the bulk value (Sung et al., 2014). One quintuple layer is approximately 1.1 nm (Sung et al., 2014), putting our materials at roughly 36 quintuple layers, albeit disordered in our case. Therefore, we would not expect surface states to significantly alter the bandstructure, and therefore, the Seebeck coefficient in our

Citation	Material	Resistivity (Ωcm)	Seebeck Coefficient (μV/K)	Details
(Horne, 1959)	Bi_2Te_3	$2.7 \times 10^{-}4$ to $1.71 \times 10^{-}3$	+160 to -145	bulk with oxide impurities
(Bottner et al., 2004)	Bi_2Te_3	$2.1 \times 10^{-}3$	-50	10 microns, sputtered
(Silva, Kaviany, and Uher, 2005)	Bi_2Te_3	$8 \times 10^{-}3$	-50	coevaporated, 1 micron
(Zou, D. Rowe, and Williams, 2002)	Bi_2Te_3	$1.29 \times 10^{-}3$ to $2.6 \times 10^{-}3$	-143 to -228	coevaporated, coevaporated, 700 nm
(Silva, Kaviany, and Uher, 2005)	Sb_2Te_3	$4 \times 10^{-}3$	+140	coevaporated, 1 micron
(Pinisetty et al., 2011)	Sb_2Te_3	x	+70 to + 365	100 and 400 nm diameters. 100 nm amorphous gave α +230 μV/K
(Baily and Emin, 2006)	Sb_2Te_3	1.11	+700, +800	amorphous, cosputtered, 1 micron
(Shi et al., 2008)	Sb_2Te_3	0.91	+125	hydrothermal synthesis
(Zou, D. Rowe, and Williams, 2002)	Sb_2Te_3	$1.04 \times 10^{-}3$ to $4.90 \times 10^{-}3$	+140 to +171	coevaporated, 700 nm

Table 3.2: Literature values of Seebeck coefficient and resistivity.

130 nm by 40 nm wires from the Seebeck coefficient in our 40 nm thick films. Our thermal simulations combined with our experimentally measured voltage support this theory. Furthermore, in a study on thin film Sb_2Te_3, it was found that while resistivity increased with decreasing film thickness (from 790 nm to 160 nm), the film thickness had little effect on the Seebeck coefficient (Rajagopalan and Ghosh, 1963), which approaches our thickness value of 40 nm.

Bi_2Te_3 and Sb_2Te_3 have been extensively studied and characterized. As with other thermoelectric materials, the Seebeck coefficient has been found to depend heavily on deposition method and deposition temperature, as well as post-deposition treatment, such as annealing (Takashiri et al., 2008; D.-H. Kim and Lee, 2006; Rashid, Cho, and Chung, 2013). Our Seebeck coefficients fall within the range of those found in the literature, a very small subset of which is shown in Table 3.2. The ranges of Seebeck coefficient for Bi_2Te_3 in this table range from +160 μV/K to -228 μV/K, depending on deposition method, percent Te, and oxide content. Our measured Bi_2Te_3 samples lie well within this range. Comparing our measured values

to any of the values in the literature is difficult, as we were not able to determine the substrate temperature during deposition. As sputterers have different working distances and different deposition powers can be used, the substrate heating would be different in each sputterer or evaporator unless substrate heating controls are used. Sb_2Te_3 in Table 3.2 ranges in Seebeck coefficient from +70 μV/K to +800 μV/K. Of note, noncrystaline Sb_2Te_3 (Ref. 7) has a higher Seebeck coefficient, 700-800 μV/K. Our samples are nearly noncrystaline, based on our XRD data (Figure 3.1). Also, in this paper, two different samples deposited in the same way had a difference in Seebeck coefficient of about 100 μV/K, showing the variability in Seebeck coefficients among similar thin film samples.

The resistivities of our materials were found by measuring cloverleaf samples in a DC hall measurement system with short time constants, and showed very small error. We found a resistivity of $2.75\times10^{-2}7.2\times10^{-5}$ Ω·cm for Bi_2Te_3 and $4.27\times10^{-1}1.1\times10^{-3}$ Ω·cm for Sb_2Te_3. A subset of literature resistivity values, shown in Table 3.2, vary between 1.71×10^{-3} Ω·cm to 8.71×10^{-3} Ω·cm. Our resistivity is higher than these values, but our films are much thinner and surface oxidation could play a significant role in decreasing conductivity. A subset of literature values (Table 3.2) show 1.11 Ω·cm to 1.04×10^{-3} Ω·cm resistivities for various thin film thicknesses and deposition methods for antimony telluride. Our values lie within the literature range, closer to the values of amorphous films. Additionally, it was found that resistivity increases with decreasing film thickness in Sb_2Te_3 (Rajagopalan and Ghosh, 1963). As our films are thinner than many in the literature, a higher resistivity, similar to ours, is expected.

We expect resistivity to be the same in our structures as in our wires, as the crystal grain sizes are small in both. Using our measured resistivity, the power factors for Sb_2Te_3 are 1.1×10^{-5}˜1.6×10^{-5} W/K^2·m and for Bi_2Te_3 $2.2 \times 10^{-5} - 3.3 \times 10^{-5}$ W/K^2·m. Using the literature values of thermal conductivity used in our thermal simulations (see previous section), this would give us zT values in the 10^{-3} - 10^{-4} range, indicating that our materials could be further optimized.

3.3 Measurement results

Measured absorption and responsivity

This section summarizes the measurements for our thermoelectric plasmonic guided mode resonance structures, with measured absorption (1-transmission-reflection) at normal (0°±1°), 5°±1° off-normal, and 10°±1° off-normal incidence (1-transmission

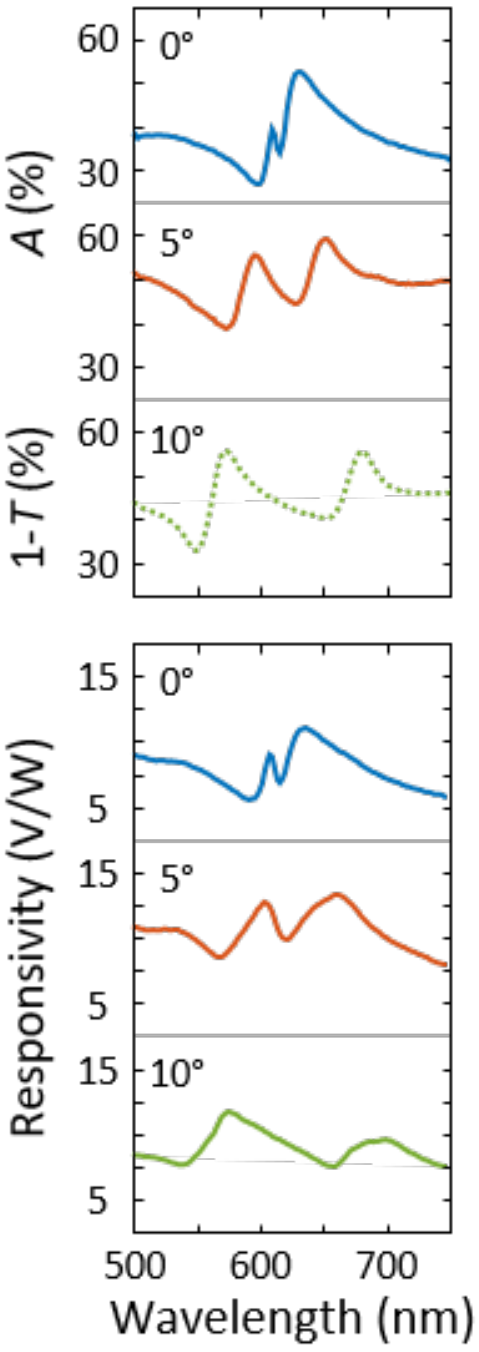

Figure 3.4: (Top) Absorption (0° and 5°) or 1 – transmission (10°) for 0°, 5°, and 10° (±1° error) incident illumination on a Bi_2Te_3 - Sb_2Te_3 structure described in the text with wire dimensions of 40 nm thick × 130 nm wide × 50 µm long. (Bottom) Responsivity for unfocused, spatially uniform illumination of the entire structure (including the pads, Figure 3.5d) with a 120 µm by 100 µm spot size at 0°, 5°, and 10° (±1° error) off-normal incidence.

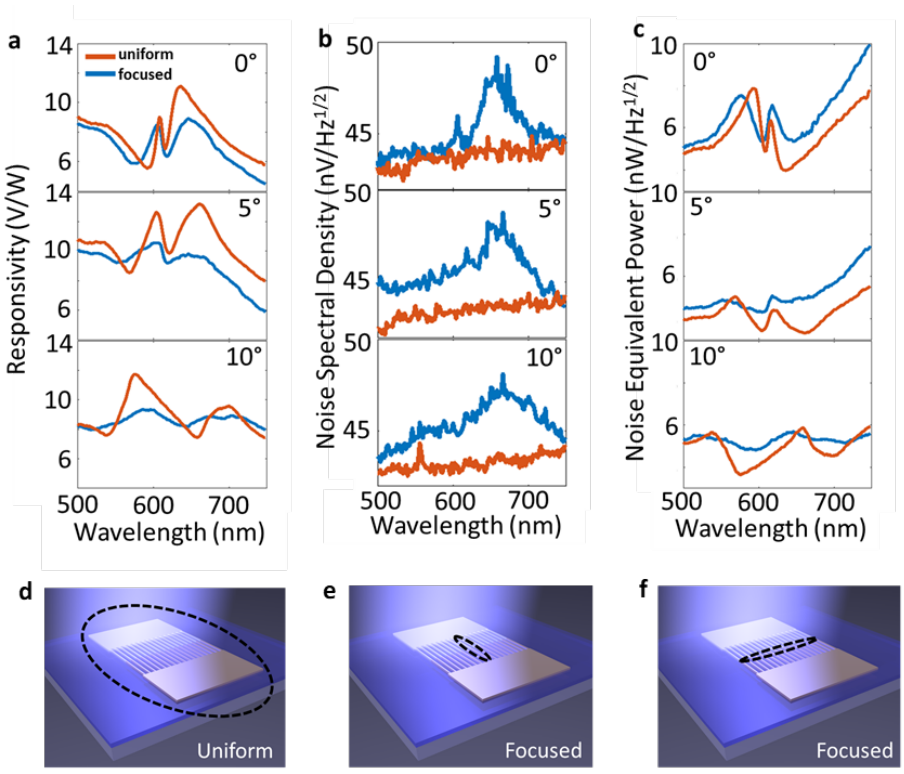

Figure 3.5: Focused versus unfocused, spatially uniform light responsivity and noise characteristics. Focused (blue, 60 µm by 5 µm spot size, **e**) and unfocused, spatially uniform (orange, 120 µm by 100 µm spot size, **d**) illumination incident on Bi_2Te_3-Sb_2Te_3 structures at given angles off normal incidence, with ±1° error. A 5× objective with numerical aperture 0.14 was used for both the focused and spatially uniform illumination data collection. **a**, The input power used to calculated responsivity in the case of uniform illumination was only the power that illuminated the wires (a 50 × 50 μm^2 area). The spatially uniform illumination spot was 120 µm by 100 µm, and completely covered the wires and pads of the structure. Noise spectral density, **b**, was measured under the power spectrum shown in Figure 3.6. Higher noise spectral density in focused light was likely due to back currents from uneven heating in the structure, discussed further in the text. **c**, Noise equivalent power was found to be lower for spatially uniform illumination than for focused illumination, due to higher responsivity values combined with lower noise values for spatially uniform illumination. **d**, Black circle illustrates uniform illumination of structure. **e**, Focused illumination used in **a-c**, Figure 3.8. **f**, Focused illumination in Figure 3.7.

shown for this case), shown in Figure 3.4 (top). Figure 3.4 (bottom) depicts the responsivity, relative to power illuminating the wire region, of a Bi_2Te_3 - Sb_2Te_3 structure completely and uniformly illuminated (pads and wires). In Figure 3.7, a

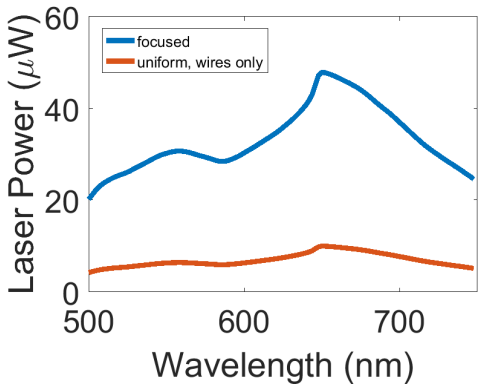

Figure 3.6: Laser power illuminating the wire region as a function of wavelength for focused illumination (blue) and uniform illumination (orange).

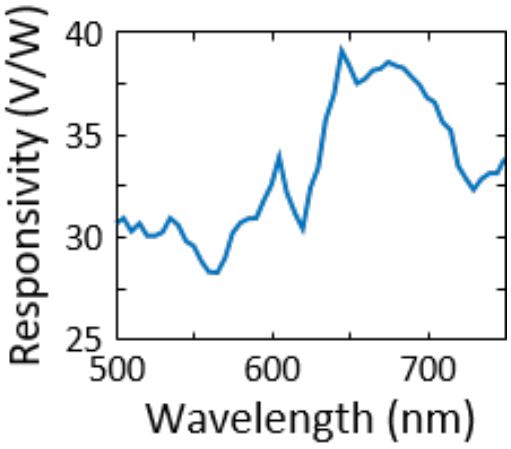

Figure 3.7: Maximum responsivity found for a structure when only the junction is illuminated (60 um by 5 um spot size, Figure 3.5f).

long, narrow beam was focused on the junction of all wires at 5°±1° off-normal incidence and represents the maximum responsivity found. The responsivity is noisier due to the sensitivity of the sample to the position of the light at the junction. Comparison of illumination configurations and alumel-chromel structure data is discussed in Figures 3.5,3.11 and in a later section. While the ratio of maximum to minimum responsivity is not large in our structures, the ratio is nearly the same as the maximum to minimum absorption ratio, suggesting that the absorption spectra largely dictates responsivity spectral shape, as demonstrated in simulations of responsivity in guided mode resonance structures in Figure 3.12. Therefore, a spectrum with a larger maximum to minimum absorption ratio would have a larger

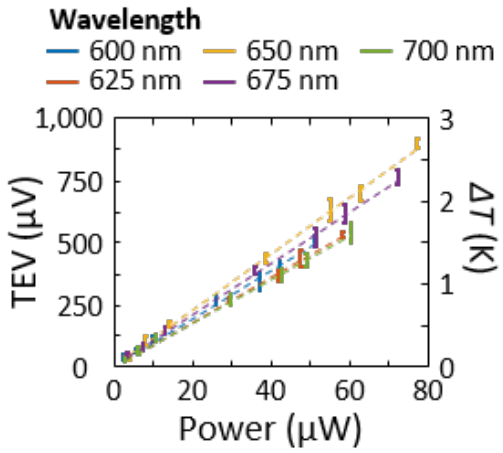

Figure 3.8: Thermoelectric voltage (TEV) dependence on incident power for a Bi_2Te_3 - Sb_2Te_3 structure at 0° (±1° error) off normal angle under focused illumination (see Figure 3.5e, and Figure 3.5a for focused responsivity spectrum). The temperature scale on the right axis corresponds to ΔT between the hot wire junctions and cold pad edges based on average measured Seebeck coefficients. We estimate that 1,000 μV would give a temperature range of a 2.8 to 3.4 K temperature rise, based on the range of Seebeck coefficients of our materials measured. Error bars are sample standard deviation of measurements.

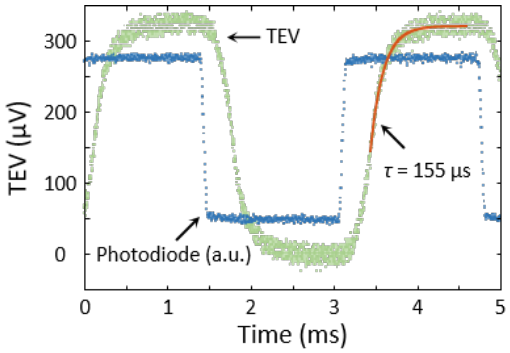

Figure 3.9: Time response of a Bi_2Te_3 - Sb_2Te_3 structure. The time constant fit line (red) plotted over the data from our thermoelectric detector (green) is measured as 155.13 s ± 3.06 s, corresponding to a 10%-90% rise time of 337 s. The response of a Si photodiode at the same chopper speed is shown in blue.

maximum to minimum responsivity ratio.

Comparison with thermal simulations and response time

We found the voltage to be linearly dependent on incident power, as shown in Figure 3.8 for a Bi_2Te_3 - Sb_2Te_3 structure under focused illumination at 5°±1° off-normal

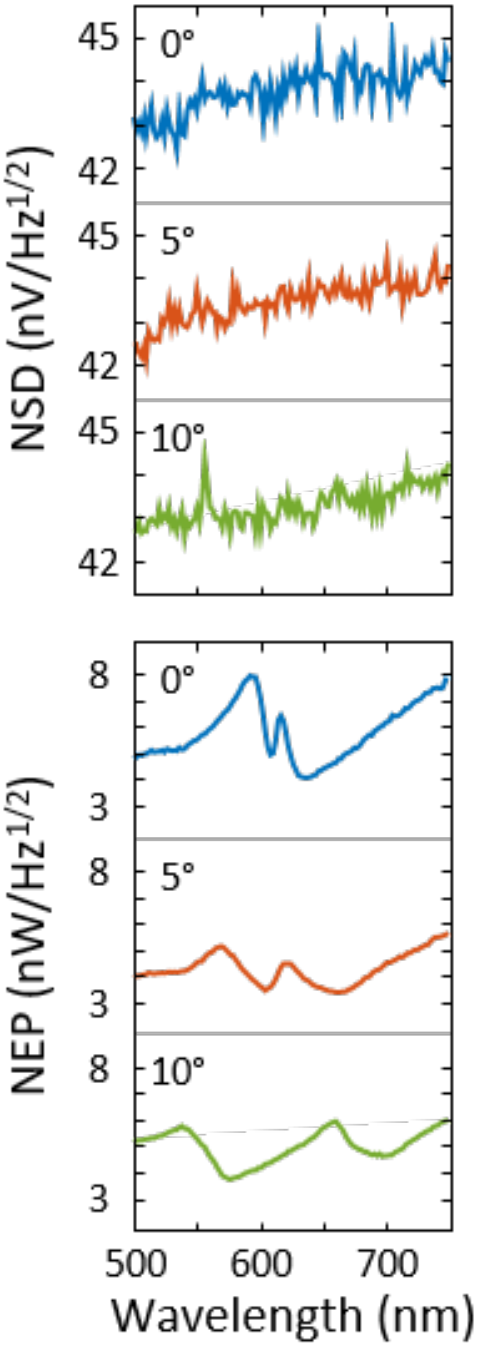

Figure 3.10: Noise spectral density (NSD) and noise equivalent power (NEP) as a function of wavelength corresponding the data shown in Figure 3.4 (bottom). All data were taken under polarized illumination with the E-field perpendicular to the wires.

incidence. The weighted root mean squared error values were 0.58 μV, 0.45 μV, 1.05 μV, 0.82 μV, and 0.74 μV for our first order polynomial fit for illumination

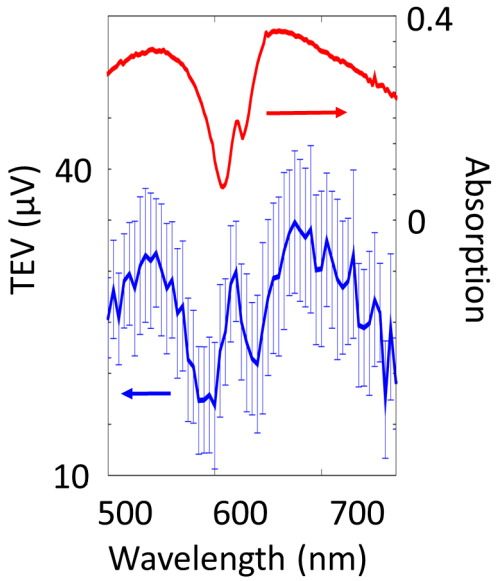

Figure 3.11: Chromel-alumel structure results. Thermoelectric potential (TEV) and absorption results for a chromel-alumel structure with the same dimensions as that of the Bi_2Te_3 - Sb_2Te_3 structures. The structure is under 70.92 W illumination, or 30.4 W/cm^2 incident power density. Data points are taken as the focused beam is moved across the junction of the detector (over a 400 μm^2 area). All data points are averaged together for a given wavelength.

wavelengths of 700 nm, 675 nm, 650 nm, 625 nm, and 600 nm, respectively. These results strongly suggest a linear dependence of TEV on incident power, which is supported by simulation (see Figure 2.37d). The temperature scale in Figure 3.8 is based on a measured average Seebeck coefficient (at room temperature) of 242 μV/K for Sb_2Te_3 and –84 μV/K for Bi_2Te_3 (see Fabrication and Materials Analysis section for details). This indicates a maximum temperature gradient ΔT of nearly 3 K, under illumination. We find a similar temperature gradient created in thermal simulations, shown in Figure 2.35. Note that the relevant ΔT is between the edge of the thermoelectric pads and the wire junctions, not the wire junctions and the simulation edge.

Measurements of the response time under chopped illumination yielded time constants of 155.13 s ± 3.06 s and 153.56 s ± 2.50 s during heat up and cool down, respectively (see Figure 3.9). This corresponds to a 10%-90% rise time of 341 s, or almost 3 kHz, which is a fast enough response for many detection and imaging applications.

Noise performance

Figure 3.10 (top) shows the noise spectral density (NSD) from the detector for an input power spectra shown in Figure 3.6. The resistance of our device is approximately 113 kΩ, giving a theoretical Johnson noise at room temperature of approximately 42 $nV/Hz^{1/2}$. Noise density detected above this level is attributed to temperature rise and unaccounted for stray capacitances or other noise sources (see next chapter for more discussion on noise measurement setups). Johnson noise provides the largest contribution to the NSD. Thus noise density could be decreased by lower device resistance through structural engineering or material selection. The noise equivalent power (NEP) is shown in Figure 3.10 (bottom). This corresponds to a detectivity of around 1×10^8 $Hz^{1/2}/W$, for a maximum of roughly 8×10^8 $Hz^{1/2}/W$ if using our maximum responsivity measurement in Figure 3.7 (see section in introduction discussing why D^* is not appropriate for most thermal detectors). In a comparison between responsivity, NSD, and NEP for focused and spatially uniform illumination conditions as a function of incident angle (shown in Figure 3.5 and discussed further in the next section), we found that the responsivity measured under spatially uniform illumination more closely matched the absorption spectra shape, and the uniform illumination had lower NEP and NSD. It is possible the higher NSD in the focused illumination case arises from shot noise or capacitor Johnson noise from back currents due to uneven heating of the thermoelectric junctions.

Comparison of illumination configuration: effect on responsivity and noise

Figure 3.5 compares responsivity and noise characteristics for a Bi_2Te_3 - Sb_2Te_3 structure under focused illumination (a 60 μm × 5 μm spot size illuminating a few single wires in the center of the array, Figure 3.5e) and spatially uniform illumination (a 120 μm × 100 μm spot size which completely uniformly illuminated the entire structure, including the pads, Figure 3.5d) over a range of angles off normal incidence, with ±1° error, and with electric field polarized perpendicular to the wires. An objective with numerical aperture 0.14 was used for all measurements. Responsivity of the spatially uniform illumination was relative to the power incident on the area of the wires only. The unfocused, spatially uniform beam leads to a higher responsivity at nearly every wavelength than that of the focused beam, and exhibited a lineshape which more closely followed the absorption lineshape in Figure 3.4 (top), and has similar shape and magnitude to our simulated values shown in Figure 3.12g-i (simulated absorption and thermal simulation) and Figure 3.12j-l (experimental absorption and thermal simulation). We believe the discrepancy between absorption

and responsivity lineshape under focused illumination arises from uneven heating of the wire array. Because our wires are very optically lossy (see the non-negligible imaginary dielectric constant in Figures 2.2, 2.4, 2.3, 2.5), light diffracted into a waveguide mode by light focused on the center wires will not propagate far, and therefore wires on the edge of the structure will not absorb (or heat up) as much as wires in the center of the array. The wires with minimal temperature gradient provide a leakage pathway for thermoelectric currents, reducing responsivity and altering responsivity spectra. Spatially uniform illumination will heat the wires more equally, largely eliminating these effects.

Leakage currents from uneven heating could cause the larger noise spectral density (NSD) under focused illumination, seen in Figure 3.5b. While this NSD is convoluted with the input laser power (shown in Figure 3.6) at each wavelength, attempts to normalize NSD to Johnson noise of a temperature rise for a given power input (using knowledge that temperature rise is linear with power input, shown in Figure 3.8) did not yield a flat NSD spectra. Therefore, excess NSD must be due to noise sources which do not vary as $T^{1/2}$, as Johnson noise does. Shot noise, on the other hand, is proportional to the magnitude of the currents in a structure, and is a possible noise source arising from these back currents if there is generation/recombination within the semiconductor materials. The NEP is lower for nearly every wavelength for the uniform illumination, and gives detectivity values in the 1-3×10^8 $Hz^{1/2}$/W range.

Comparison with measured alumel-chromel device

Thermoelectric potential (TEV) and absorption results for a chromel - alumel structure with the same dimensions as that of the bismuth telluride structures is shown in Figure 3.11. The structure is shown under 70.92 W illumination, or 30.4 W/cm^2 incident power density. The focused beam is raster scanned across the wire junction region of the detector, and at each location, a data point is taken (over a 400 μm^2 area). All data points are averaged together.

3.4 Best possible performance with state-of-the-art materials for current design

Large responsivity and low noise are needed to have high detectivity. Larger responsivity arises from a higher Seebeck coefficient, α, and a larger temperature gradient (therefore, smaller thermal conductivity, k). Lower Johnson noise (the primary noise source in our structure, shown in Figure 3.10 (top)), comes from lower

resistance, and therefore lower resistivity, ρ. Thus, detectivity can be improved with a higher thermoelectric figure of merit, $zT = \frac{\alpha^2 T}{\rho k}$, material. At room temperature, one of the best p-type materials is a BiSbTe alloy which has a zT of 1.2 at room temperature (Poudel, Hao, Ma, Lan, Minnich, Yu, X. Yan, D. Wang, Muto, Vashaee, et al., 2008b). One of the best performing n-type materials is a PbSeTe quantum dot superlattice material with a zT of 1.3-1.6 at room temperature (T. C. Harman et al., 2002). The resistivity of the high zT materials are 1.71 x10-3 Ωcm (n-type) and 8.00 x10-4 Ωcm (p-type), the thermal conductivities are 0.58 W/m·K (n-type) and 1.1 W/m·K (p-type), and the Seebeck coefficients are -219 μV/K (n-type) and 185 μV/K (p-type). While we did not measure the thermal conductivity of our materials, assuming bulk values (D. M. Rowe and Bhandari, 1995) of thermal conductivity for our Bi_2Te_3 (2.05 W/m·K) and Sb_2Te_3 (3.54 W/m·K), we can see that the overall thermal conductivity with the high zT materials would be decreased by a factor of 4. While our thermal simulations were able to explain experimental results reasonably well (compare Figures 3.12g-l and Figure 3.4 (bottom)), it is known that small crystal grain sizes can decrease thermal conductivity in thin film BiTe-based materials(Takashiri et al., 2008). From our XPS data (Figure 3.1), we can see that the grain sizes must be small. This indicates that our thermal conductivity may not be greatly improved by using the state-of-the-art materials above. In results from thermal simulations we can see that using state-of-the-art n-type material, shown in Figures 3.12d-f, versus our n-type Bi_2Te_3 material, shown in Figures 3.12g-i, we get 40% increase in responsivity. Our structures depend both on the thermal conductivity of the thermoelectric material, and the thermal conductivity of the substrate. Fortunately, suspended low-stress SiN_x membranes(Ftouni et al., 2015) and SiO_2 can have thermal conductivities lower than or similar to the thermoelectric materials we study. To remove the thermal conductivity contribution from the substrate, the substrate could be etched from beneath the wires, although this would affect the optical absorption characteristics.

The Seebeck coefficient for the large zT n-type material are larger than ours by a factor of 2.6 (-219 μV/K versus an average of -84 μV/K), and the Seebeck coefficients for the large zT p-type material were smaller than our materials by a factor of 1.3 (185 μV/K versus an average of 242 μV/K). This gives an overall Seebeck coefficient of 404 μV/K for the optimal materials, versus 326 μV/K for our materials, or a factor of 1.24 increase in Seebeck coefficient. The resistivity of the large zT n-type material decreases by a factor of 16 over our n-type material resistivity (1.71×10-3 Ω·cm versus 2.75×10-2 Ω·cm), and the resistivity of the large zT p-type material

decreases by a factor of 534 over our p-type material resistivity (8.00×10-4 Ω·cm versus 4.27×10-1 Ω·cm). We can treat our system as two resistors in series, one resistor composed of n-type material and the other composed of p-type material. Then, assuming a symmetric system, the total resistance is proportional to the sum of the resistivities of the two materials. The sum of the resistivities of the Bi_2Te_3 and Sb_2Te_3 materials in our paper is 4.55×10-1 Ω·cm, and the sum of the resistivities of the state-of-the-art materials is 2.51×10-3 Ω·cm. Thus, resistance decreases by a factor of about 181 from our materials to the state-of-the-art. While nanostructuring could change the material properties of the high zT materials, an approximation for the expected detectivity increase can still be found.

Based on our simulations, a factor of 4 decrease in thermal conductivity in our thermoelectric materials will produce a factor of 1.4 increase in temperature difference between the hot and cold end of our device. Coupled with a factor of 1.24 increase in Seebeck coefficient, we can expect the responsivity to increase by a factor of 1.7 using the state-of-the-art thermoelectric materials. Noise spectral density depends on the square root of resistance (assuming Johnson noise dominates), so decreasing resistance by a factor of 181 will lead to a decrease in noise by about a factor of 13. Therefore, the detectivity can be increased by around 1.7 × 13 = 22 times by using state-of-the-art thermoelectric materials.

3.5 Why D^* is not a valid FOM in our structures and other thermal detectors

As described in (Datskos and Lavrik, 2003), specific detectivity (D^*) was a figure of merit designed for comparing quantum detectors, independent of detector size. Quantum detectors have detectivity which scales with the square root of the detector area. As we explain below, detectivity for our structure does not scale with the square root of the detector area, but with a more complicated function of the width of detector and length of the legs. Thus, our size-independent detectivity figure of merit would have different units than that of quantum detectors, so we hesitate to present this value. Page 351 of (Datskos and Lavrik, 2003) states:

> *"It should be noted that the definition of specific detectivity, D*, was originally proposed for quantum detectors, in which the noise power is always proportional to the detector area and noise signal (V or I) is proportional to the square root of the area. However, the noise in thermal IR detectors does not always obey this scaling trend. In fact, neither temperature fluctuations nor thermo-mechanical noise (see the*

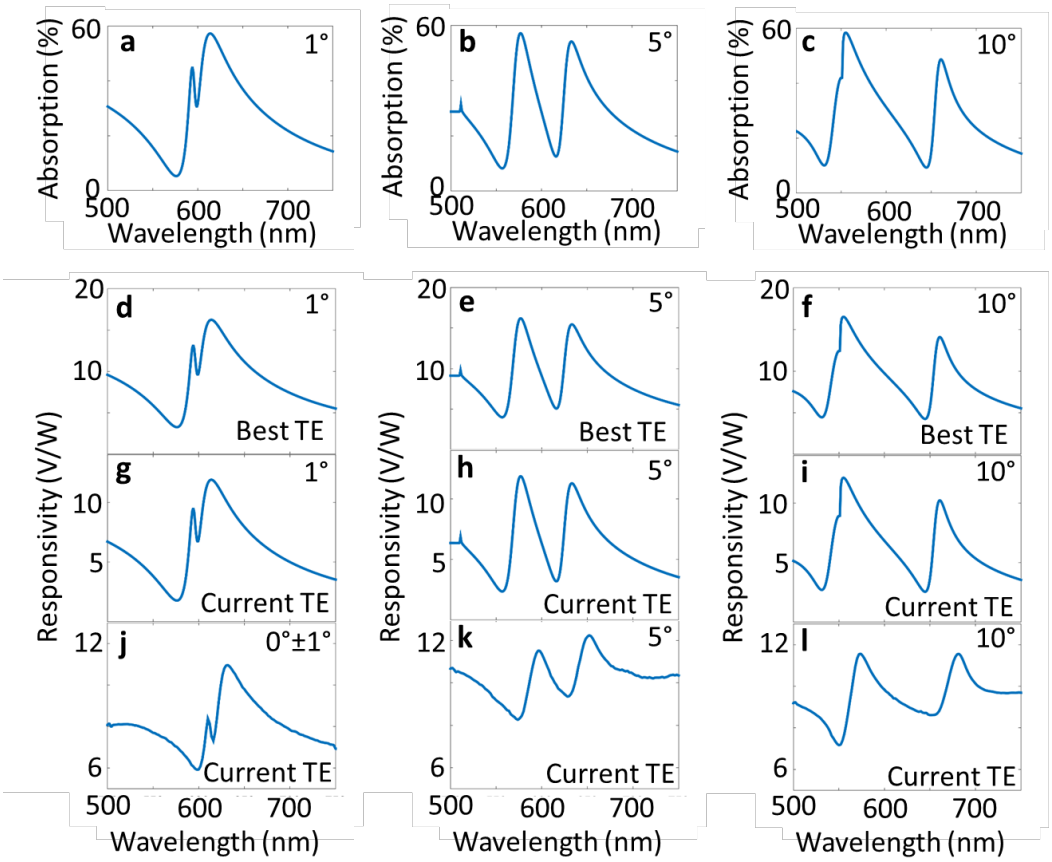

Figure 3.12: **a-c**, Simulated absorption spectra for wires of the dimensions of the experimental Bi_2Te_3-Sb_2Te_3 structures at 1, 5, and 10 degrees off normal incidence. **d-l**, Responsivity calculated from thermal simulations using absorption to guide power input at different angles off normal incidence. 20% absorption in the pads was assumed independent of wavelength. Simulated absorption from **a-c**, thermal properties and Seebeck coefficient of state-of-the-art thermoelectric materials, given in the text, were used in **d-f**. Simulated absorption from **a-c**, thermal properties given in the previous chapter and average Seebeck coefficient of our materials (242 µV/K for Sb2Te3 and -84 µV/K for Bi2Te3, see Supplementary Note 8) was used in **g-i** to calculate responsivity. Experimental absorption from Figure **??**(top), thermal properties from the previous chapter and average Seebeck coefficient of our materials (above) was used in **j-l** to calculated responsivity.

next section) scales up with the detector area. Therefore, D should be very cautiously interpreted when applied to thermal IR detector. In fact, D* tends to overestimate the performance of larger absorbing area thermal detectors and underestimates the performance of smaller ones."*

We therefore have deemed it appropriate to include only NEP and D calculations, for which (Datskos and Lavrik, 2003), p. 351 remarks:

> *"The parameter NEP is generally sufficient to evaluate and compare the performance of single (spot) IR detectors by predicting the minimum power."*

By definition, $D^* = \sqrt{A} \cdot D$, where D^* is selective detectivity, A is detector area, and D is detectivity. Detectivity is defined as $D = \frac{1}{NEP}$, where NEP, or noise equivalent power, is defined as $NEP = \frac{NSD}{r}$, where r is responsivity, or the output voltage over the input power, and NSD is the noise spectral density, with a Johnson noise floor of $NSD_{Johnson} = \sqrt{4Rk_bT}$, where R is the structure resistance, k_b is the Boltzmann constant, and T is temperature in Kelvin. In the following calculations, we ignore shot noise from emf-induced currents, and examine the minimum D that can be found assuming only Johnson noise. This is a reasonable approximation, given that our experimental NSD is very close to the Johnson noise limit (Figure 3.10 (top) and caption), indicating Johnson noise is the dominant noise source. The temperature differences between the pad edges and wire centers should be similar between a thermopiled structure (Figure 2.39), and our non-thermopiled structure (Figure 2.23), indicating we can study r, D, and D^* only as a function of resistance and dimension.

Using the relations above, we find

$$D = \frac{r}{NSD} = \frac{r_1(l)}{\sqrt{\frac{4R}{n}k_bT}} \sim \sqrt{\frac{n}{R}}r_1(l), \tag{3.1}$$

for a guided mode resonance wire structure shown in Figure 2.23 with n wires in a parallel (non-thermopile) configuration, where R is the resistance of a single wire (so $R_{total} = \frac{R}{n}$), and $r_1(l)$ is the responsivity of a single wire. We assume that $r_1(l)$, for wires of fixed cross-sectional area and with fixed, uniform thermal conductivity, is some function of the wire length (See Figure 2.37e, for example). Assuming the majority of resistance comes from the wires (not the pads), and the cross-sectional area of the wires are constant and resistivity is the same for all wires, $R \sim l$, where l is the length of the wire. Thus, $D \sim \sqrt{\frac{n}{l}}r_1(l)$. The total detector area of the structure (counting the wire area only) is approximately $A = nlp$, where p is the pitch of the wires. Then,

$$D^* \sim \sqrt{nlp} \cdot \sqrt{\frac{n}{l}} r_1(l) = \frac{npr_1(l)}{p^{1/2}} = W \cdot \frac{r_1(l)}{p^{1/2}}, \tag{3.2}$$

where W is structure width. W and $r_1(l)$ are independent of one another and will not cancel in general. Therefore, D* will not give a detection figure of merit independent of area and is not a good metric for our structures. Other thermal detectors have similar problems using D^* as a figure of merit(Datskos and Lavrik, 2003).

For a thermopiled device of n wires, where the wires are in series, i.e. total resistance $R_{total} = nR$, the detectivity (D_{TP}) is

$$D_{TP} = \frac{r_{TP}}{NSD} = \frac{nr_1(l)}{\sqrt{4nRk_bT}} \sim \frac{nr_1(l)}{\sqrt{nR}} = \sqrt{\frac{n}{R}} r_1(l) \sim \sqrt{\frac{n}{l}} r_1(l), \tag{3.3}$$

which is the same as detectivity for a thermocouple (Equation 3.1), again neglecting contributions of the pads to resistance. Since the detector area is the same for thermopiled and non-thermopiled structures, then D^* will also be the same.

As discussed in (Datskos and Lavrik, 2003), there exists other figures of merit for thermal detectors. For instance, noise equivalent temperature difference (NETD) is common in bolometric detectors, or other IR focal plane arrays. NETD is defined as "the temperature of a target above (or below) the background temperature that produces a signal in the detector equal to the rms detector noise." (Datskos and Lavrik, 2003). This figure of merit is useful as room-temperature blackbody radiation is noise in IR detectors, unlike in visible detectors. Another figure of merit is the minimum resolvable temperature difference (MRTD) which characterizes the spatial resolution and temperature sensitivity of IR detectors. Another figure of merit, D^{**}, normalizes the detectivity by the focal ratio.

3.6 Comparison with bolometers

There exists an enormous number of different light-detecting technologies, from semiconductor photodiodes, to photomultiplier tubes, to Golay cells, to photoreceptor cells in our retina, just to list a small fraction of available detectors. Each technology has its advantages and disadvantages, and when choosing a detector, a number of factors must be taken into account, including the wavelengths and bandwidth of interest, noise equivalent power or other relevant detection efficiency

figure of merit, the sensitivity range (the ability to withstand Watts of laser power versus detect at the single photon level), the speed, the suitability of the detector to its environment (radiation, temperature, mechanical vibrations), the lifetime of the detector, the power consumption, the weight, the need for cooling, the cost or availability of materials, the fabrication difficulties, etc... No detector is ideal in all situations. For instance, silicon photodiodes work well for visible light at room temperature at the nano or micro Watt level and are inexpensive, but do not work well in the infrared regime below their bandgap, or at high temperatures including under kWatt illumination, and will degrade under radiation. Photomultiplier tubes and avalanche photodiodes can work well for detecting single photons, but conversely can be permanently damaged when exposed to room light levels.

Without a specific application in mind, comparison between technologies is akin to asking the question, "In a fight, would a shark or a polar bear win?", as clearly, the environment of the fight matters. Fifty meters underwater, a shark is the clear winner. On land, a polar bear would be victorious. Similarly, comparing a photomulitiplier tube for single photon detection to a thermoelectric detector designed for kWatt detection doesn't make sense without a specific application in mind, as they are well suited for completely different applications.

Choosing a similar technology, we will compare our thermoelectric nanophotonic structure with a bolometer based on the criterion listed above. A bolometer is composed of a material which has a well characterized resistance change with temperature, so that when incident light impinges upon the material, the temperature rises and the corresponding resistance change is measured by monitoring the voltage change of a small current passing through the resistor. Both bolometric and thermoelectric light detectors are most sensitive when size is minimized and a large thermal resistance exists between the hot region and reservoir (or "cold side" of the thermoelectric element). The wavelength selectivity for both technologies is determined in similar ways, via geometry, or primarily non-material-dependent properties (in contrast to photodiodes, for instance, which have wavelength selectivity dictated by bandgap). Materials selection of the sensing region is important in both cases, however. For more sensitive bolometers, materials with higher temperature coefficient of resistance is desired. Thermoelectrics require materials with high Seebeck coefficients. Thermopiles operate most efficiently with two high zT materials of opposite sign Seebeck coefficients, but this is not strictly necessary. A high zT thermoelectric material could be paired with a low zT material with lower

resistance, such as a metal, which would decrease Johnson noise, potentially at the expense of lower temperature drop due to the higher thermal conductivity of metals. Both bolometers and thermocouples additionally require materials with stable characteristics to prevent drift in accuracy over time. The dominant noise source in each application should be Johnson noise from the resistive character of the sensing elements, as well as 1/f noise at low frequencies. Noise from amplifiers will also contribute, and bolometers will have an additional noise source from the source of injected current to extract voltage. In both cases, decreasing the temperature of the device will decrease Johnson noise, but care must be taken that the sensing materials still operate in lower temperature regimes. For instance, the Seebeck coefficient of bismuth telluride drops at lower temperatures, and another thermoelectric material must be used.

In terms of power requirements, from a theoretical perspective, thermoelectrics should require less power. This is due to the thermoelectric materials producing a voltage via the thermoelectric emf, which can be on the order of hundreds of microvolts per Kelvin. This signal may or may not need to be amplified to be sensed. Bolometers, on the other hand, require external work done in the form of a supplied current in order to convert resistance changes into a measurable voltage signal. Likely a small current must be used to prevent significant Joule heating, so an amplifier also must be used to detect the signal. Therefore, a thermopile detector should have lower voltage requirements. In reality, often thermopile detectors have an internal silicon diode which monitors the room or reservoir temperature (as thermoelectrics can only give information about temperature differences, not absolute temperature). The silicon diode has a resistance change with temperature, so a current must be supplied to monitor this as well, but in principle this does not need to be continuously monitored, but checked at intervals, depending on the required parameters.

Bolometers have been shown to detect single photons with near unity efficiency (Lita, Miller, and Nam, 2008). In principle, assuming 100% light absorption, thermopiles should be able to sense single photons with some non-negligible efficiency, dependent upon the efficiency of low temperature thermoelectric materials at the low temperatures necessary for single photon detection via Johnson noise reduction. Additionally, the thermoelectric effect relies only on the electron temperature, not the phonon temperature, so at lower temperatures where electrons are more easily decoupled from phonons, perhaps more efficient detection can be achieved, similar

to hot electron bolometers. More research is needed to determine the experimental and theoretical sensitivity limit of thermoelectric nanophotonics.

3.7 Outlook, improving performance, and potential applications

The responsivity and detectivity of these structures could be increased through thermopiling, optimizing the thermoelectric materials, measuring in vacuum to eliminate convective loss, or suspending the wires to eliminate conductive losses to the substrate. Focusing on material optimization alone as an example, responsivity will increase with a higher Seebeck coefficient and lower thermal conductivity (k). The noise floor can be decreased with lower resistivity (ρ). Therefore, detectivity can be increased using a material with a larger thermoelectric figure of merit, $zT = (\alpha^2 T)/\rho k$. For example, high room temperature zT n and p-type materials, such as a p-type BiSbTe alloy (Poudel, Hao, Ma, Lan, Minnich, Yu, X. Yan, D. Wang, Muto, Vashaee, et al., 2008a) with room temperature zT = 1.2, and n-type PbSeTe-based superlattice structure (T. Harman et al., 2002) with zT = 1.6 can be used. Alone, the increased Seebeck coefficient of these materials (~25% combined increase over our structure) would increase responsivity and detectivity by roughly 25%. Using these state-of-the-art thermoelectric materials in our structure, would lead to a factor of 1.7 and 22 overall increase in responsivity and detectivity, respectively (see earlier section).

Thermopiling would further boost device responsivity, shown schematically in Figure 2.39 with simulated responsivity as a function of number of wires in the thermopile shown in Figure 2.37b. As we observed in our guided mode resonance structures, focusing light using a far field lens at all thermoelectric junctions maximized responsivity (Figure 3.7). Light can also be focused onto a thermoelectric junction by using plasmonic nanophotonic structures (Coppens et al., 2013a) designed to maximize the electric field inside the thermoelectric material, as illustrated by the plasmonic bow-tie antenna shown in Figures 2.17, 2.18, 2.19, 2.21. Guided mode resonance structures are highly angle sensitive, whereas relatively angle-insensitive performance can be achieved using e.g., 'perfect absorber' antenna structures (Alaee et al., 2012; Aydin et al., 2011; K. Chen, Adato, and Altug, 2012; Liu et al., 2010b) or split-ring resonators (Landy et al., 2008) that excite a thin thermoelectric junction like those shown in Figures 2.1, 2.6, 2.7, 2.9 and Figures 2.11, 2.12, 2.15, respectively. The perfect absorbing structures and split-ring resonator absorbers also exhibit 10-20 times lower noise equivalent power than the guided mode resonance wire structures, as discussed.

While conventional photodiodes exhibit higher detectivity and response times in the visible regime, resonant thermoelectric light detecting structures have two primary advantageous features. First, thermoelectric resonant structures are band-gap insensitive and have shown potential as room temperature infrared light detectors, as an alternative to super-cooled photodiodes or bolometers. Second, as we have shown, resonant thermoelectric structures can have response times 100 times faster than previously reported thermoelectric detectors made from high zT materials arising from the smaller heat capacity of resonant thermoelectric structures resulting from their large absorption cross-section (see previous section). Additionally, these structures combine responsivity with wavelength selectivity, enabling easier fabrication. It may be possible to design very compact resonant thermoelectric structures that exhibit sufficiently large thermal gradients over short distances (1-5 μm) such as the one illustrated in Figure 2.19, which may make it possible to shrink thermoelectric sensors to a scale more comparable to conventional camera pixel sizes of $10\ \mu m^2$.

Using nanophotonic designs to better focus the electric field on an as-small-as-possible section of the thermoelectric junction (Figure 2.18) could improve performance by maximizing the temperature difference between the hot and cold end of the thermoelectric elements. Suspending the junction to minimize heat conducted away by the substrate, combined with cooling the "cold" ends of the thermoelectric materials by putting high thermal conductivity materials near the "cold" regions (Figure 2.7), would increase responsivity by increasing the temperature difference within the thermoelectric structures. Shrinking devices will additionally decrease Johnson noise in resonant thermoelectric structures, thereby decreasing NEP. We note that thermal design of parallel-connected thermoelectric junctions should minimize uneven junction heating, which can cause internal currents that waste input energy. In general, careful consideration of matching the optical power absorbed to the thermal impedance will be required to optimize thermoelectric nanophotonic structure performance.

Chapter 4

NANOSCALE TEMPERATURE MEASUREMENTS WITH NOISE THERMOMETRY AND THE THERMOELECTRIC EFFECT - LOW NOISE DESIGN

4.1 Nanophotonic temperature measurement review

Measuring the temperature of nanostructures is important for a number of applications. There has been an increase in interest in determining the temperature of nanophotonic structures under illumination, specifically those illuminated structures which increase chemical reaction rates. It is desirable to elucidate the exact mechanisms for the rate increase, such as temperature rises versus hot electron generation (Golubev et al., 2018). Additionally, measuring temperature in nanoscale structures or gaps is of interest for thermophotovoltaic devices (Lenert et al., 2014), thermal emitters (Yeng et al., 2012), or in near-field heat transfer studies and applications (Yang, Narayanaswamy, and G. Chen, 2005). Other applications lie in measuring the heat capacity or thermal conductivity on the nanoscale, which has proven challenging. Finally, one may want to study electronic property changes under nanophotonic heating or illumination, which requires temperature measurements (Schoenlein et al., 1987).

The challenge is finding a way to measure the temperature of nanophotonic structures without affecting their nanophotonic properties. A very large and diverse number of creative techniques exist for measuring the temperature of nanophotonic structures. Raman thermometry (e.g. (Davis et al., 1993)) determines the temperature of a structure by examining shifts in Raman peaks or by comparing the magnitude of Stokes and anti-Stokes peaks. Thermoreflectance (e.g. (Antonelli et al., 2006)) measurements use changes in reflection of light from a surface under different thermal conditions to extract temperature. Fourier transform infrared microscopy (FTIR) uses a measurement of the blackbody spectrum of a material to determine temperature. By measuring infrared wavelengths, however, the spot size is diffraction limited and can be quite large, on the order of 10's of microns. The diffraction limit can be overcome by coupling an FTIR to an atomic force microscope (AFM) to make Photothermal infrared (PTIR) measurements (Katzenmeyer et al., 2015; Hammiche et al., 1999). A different AFM-based technique is scanning ther-

mal microscopy (SThM) (K. Kim et al., 2012), which has a thermocouple junction at the tip of an AFM probe. This can be used for nanophotonic heating measurements (Grajower et al., 2015). The drawback of this measurement technique is that the probe is approximately 10 nm from the surface and can potentially interfere with the photonics of the system. Another method is temperature-dependent photoluminescence measurements (Coppens et al., 2013b), where the substrate a nanostructure sits on has temperature-dependent photoluminescence. Resistive thermometry is also a temperature measurement method, whereby either the resistance of a material is carefully measured with temperature (Herzog, Knight, and Natelson, 2014), or a platinum resistive thermometer is placed on the material, and the resistance of this is measured to extract temperature. A number of other interesting methods exist, including but by no means limited to (Guillaume Baffou et al., 2012; Carlson, Khan, and Richardson, 2011; G Baffou et al., 2009; Ward et al., 2011; Rosencwaig et al., 1985).

The major drawback of many of these methods is the need for either an expensive microscope or laser setup, and/or the need for excessive calibration for each device (as in the case of resistive thermometry). Additionally, the temperature must be extrapolated from a measurement method which introduces a number of error sources, such as laser power. We propose noise thermometry, a primary temperature measurement method, (see introduction for noise thermometry overview), in conjunction with the thermoelectric effect as an alternative measurement technique which only requires electrical leads to the nanophotonic device, does not involve coupling and aligning an extra laser, has minimal calibration, and measures the temperature of the electrons themselves.

4.2 Measurement plan

The aim of this project is to measure the temperature at the hot spot of a nanophotonic device. Others have used noise thermometry in a graphene bolometer application (Efetov et al., 2018), but they required a thermal model to determine the temperature of the hot spot in the device relative to the edges (ΔT), which increases uncertainty. In our method, we directly measure the hot-spot absolute temperature. Figure 4.1 shows an SEM of a typical structure we extract the temperature of. In order to determine the temperature at the "hot spot" of the device, or the junction of the wires, two steps must be performed. First, the Seebeck coefficient of the Bi_2Te_3 must be determined. From an earlier chapter, we know that Seebeck coefficient of Bi_2Te_3 can vary from -228 to +160 μV/K depending on deposition conditions, so estimating the

Seebeck coefficient from the literature will almost certainly yield the wrong result, possibly including the wrong sign. To measure the Seebeck coefficient, we measure the noise temperature across each lead on the Bi_2Te_3 device, and measure the voltage between each set of leads. We know $\Delta V = \alpha \Delta T$, where α is the Seebeck coefficient, so by measuring ΔV and ΔT, we can extract α. With α, we can proceed to step 2. Here, we measure the noise temperature at the lead set on the Bi_2Te_3 closest to the hot spot, and the voltage difference between these leads and the platinum leads shown in the picture. Thus, with ΔV and α (we assume the Seebeck coefficient of Pt is much smaller than that of Bi_2Te_3 and can be neglected (Moore and Graves, 1973)), we can extract ΔT_{hot}, the temperature difference between the center of the wires and the Bi_2Te_3 lead closest to the hot spot. The noise temperature gives us the *absolute* temperature at the leads closest to the hot spot on the Bi_2Te_3 side, and this temperature plus ΔT_{hot} gives the *absolute* temperature of the hot spot.

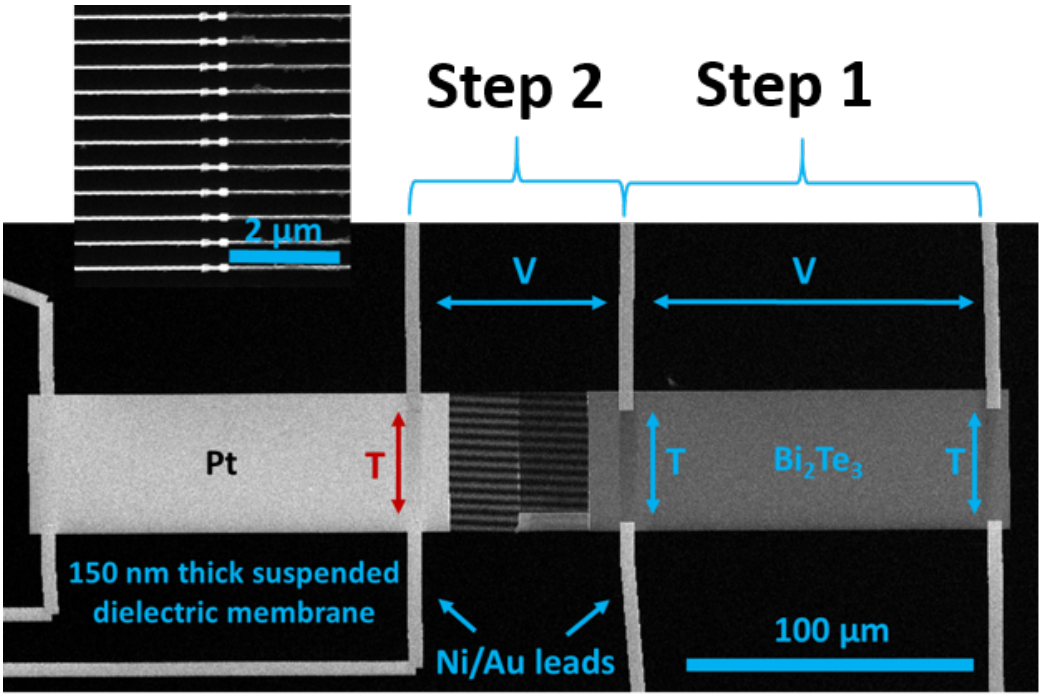

Figure 4.1: Measurement plan outline. The inset shows the wire junctions. The laser broadly illuminates the wires, and the junction should be the hottest point in the structure. In principle, because of the optical properties of this structure discussed in earlier chapters, the entire structure could be illuminated by the laser.

This method gives the added benefit of being able to monitor the temperature changes across the device due to thermoelectric currents in the device. In addition to performing noise thermometry across the device leads, we also monitor the

resistance in the same locations using a bandpass filter and lock-in at frequencies outside of the noise thermometry measurement window. This allows us to extract more precisely the change in noise due to change in temperature by eliminating noise changes due to changes in resistance.

This method requires collection of data from at least three leads at the same time. The amplifing circuit will be discussed in the next chapter. The DAQ we used at first was the National Instruments USB 4431 with 4 channels of simultaneous 24-bit 100 kSamples/s sampling. This DAQ was used for noise measurements in the previous chapter; however, we found that with the slower data rate, to collect enough data to get small enough error bars on our temperature measurements, we had to collect continuously (with breaks to save data due to memory constraints) for about an hour. During this time, the temperature in the room or on the circuit board could drift by several Kelvin, hurting our temperature resolution. We upgraded to the 8-channel National Instruments USB 6366, which has 2 MSamples/s sampling rate with 16-bit resolution. Temperature resolution will be discussed in the next chapter.

The USB 6366 only has DC coupling, so AC coupling for each channel (in order to get better resolution in case of DC offsets), composed of a high pass filter in a Pomona box consisting of a 100 kΩ resistor and a 10 μF capacitor. Shorting the DAQ, we measured a noise of approximately 38 nV/$\sqrt{\text{Hz}}$.

4.3 Fabrication

Fabrication of this device was similar to the device studied above. The same PECVD and electron beam lithography systems were used. Instead of Sb_2Te_3, Pt was used deposited with an electron beam evaporator, with a 15 nm Ni adhesion layer, as Ni makes an ohmic contact to Bi_2Te_3. Bi_2Te_3 was cosputtered this time, with 24 W RF power for Bi and 40 W RF power for Te, sputtered at 3 mTorr in Ar. Leads to the samples were composed of 15 nm of Ni (for Ohmic contact) and 120 nm of Au.

4.4 Design considerations for circuit board

In designing the circuit board, several important factors had to be considered. First, the choice of operational amplifier depended on the source resistance, or the resistance of the thermoelectric nanostructure we were measuring the noise temperature of. Second, the thermal heating of the circuit board will cause drift in the noise of the opamps over time. These two points will be discussed futher in the following sections. Much of the following information is sourced from (Motchenbacher and Connelly, 1992) and (Kay, 2012). An alternative method for

noise measurement of high resistance sources is to use an inductor (Fong and KC Schwab, 2012; Efetov et al., 2018). We opted not to, as our source resistances were low enough to not have significant rolloff from stray capacitances. Higher fidelity noise measurements can be obtained via techniques NIST is exploring (e.g. (Qu et al., 2013)).

Opamp considerations

There are two important noise figures in opamps: voltage noise, and current noise. Voltage noise arrises from various resistances within the opamp, and the current noise arises from shot noise in the transistors (see the the noise theory section in the introduction for more general information). For a source impedance, Z_s, which includes the source resistance along with any coupling capacitors or inductors, the equation for noise power relative to the ouput of the opamp (not including the data acquisition device noise) is

$$E_0^2 = 4k_bT_sRe[Z_s]\left(\frac{R_2}{R_1}+1\right)^2 + (i_n|Z_s|)^2\left(\frac{R_2}{R_1}+1\right)^2 + e_n^2\left(\frac{R_2}{R_1}+1\right)^2 + 4k_bR_2T_{R_2} + (i_n|R_2|)^2 + 4k_bR_1T_{R_1}\left(\frac{R_2}{R_1}\right)^2, \tag{4.1}$$

where the first term is the Johnson noise of the source impedance (note that this term only depends on the real part of the source impedence) multiplied by the gain of the non-inverting amplifier, the second term is the noise in the source impedance generated by the current noise of the opamp (note that this term depends on the magnitude of the impedance, which is not stated in most opamp data sheets), the third term is the voltage noise of the opamp, the fourth term is the Johnson noise of the larger gain resistor, the fifth term is from the noise current of the opamp through the larger gain resistor, and the last term is the Johnson noise of the smaller gain resistor, multiplied by the gain of the inverting input. Note that in the noise sources from the gain resistors, the real part of the impedance is used, with the exception of the fifth term which uses the magnitude of the impedance (useful if you have a capacitor in parallel, for instance).

Clearly, the noise of a complicated circuit can be difficult to calculate, which is why keeping a circuit simple is important. Using eq. 4.1, we can determine the best opamp for our source resistance. Broadly, we want to have the noise from our source, or the first term in Equation 4.1 dominate the other noise sources. If our source resistance is large, e_n can be overcome, but i_n begins to play a role,

and the second term can become quite large. If our source resistance is small, e_n is more important, as the second term will be quite small. Unfortunately, there is not, to our knowledge, an opamp with low enough voltage and current noise to be used for all applications. For small source resistance, the AD797 and OPA211 from Analog Devices and Texas Instruments, respectively, have excellent voltage noise, near 1 nV/$\sqrt{\text{Hz}}$. However, their current noises are relatively large, around 2 pA/$\sqrt{\text{Hz}}$, making them unsuitable for applications with source resistances larger than approximately 2kΩ. The AD745 and OPA827, again from Analog Devices and Texas Instruments, respectively, have fairly low voltage noises (2.9 nV/$\sqrt{\text{Hz}}$ and 3.8 /$\sqrt{\text{Hz}}$, respectively, though not nearly as low as the AD797 and OPA211, but have extremely low current noise (6.9 fA/$\sqrt{\text{Hz}}$ and 2.2 fA/$\sqrt{\text{Hz}}$, respectively). Thus, for low source resistances, the e_n of these opamps will dominate, but for high source resistances, the e_n will be negligible, as will the i_n. Because the resistances of our sources are several thousand kΩ's, we chose the OPA827. The added benefit of the OPA827 over the AD745 are its well-characterized noise specs and its stability and lack of need for external compensation capacitors. The AD745 has lower voltage noise, but has a more complicated and harder to characterize noise profile because of the added external capacitance.

While the ideal circuit configuration would be to have R_s grounded on one side and connected directly to the non-inverting input on the other side, this was not possible in our circuit. For one, directly grounding one side of R_s would short circuit any thermoelectric voltage generated under illumination. Adding a large resistor between R_s and ground (shown as R_g in Fig. 4.2) would solve the problem of preserving generated thermoelectric voltage, however, this large resistor Johnson noise would dominate noise from R_s. To correct for this, C_2 is added parallel to R_g in Fig. 4.2. This "looks" like a short circuit to ground at higher frequencies where we measure noise, but acts like an open circuit at lower frequencies where we chop the laser, allowing us to extract the thermoelectric voltage while still seeing the noise. R_3 and C_1 in Fig. 4.2 AC couple to the non-inverting input opamp, which serves two purposes. The first is to avoid DC offsets which drive the opamp close to the rails (a +/- 5 V supply is used for thermal management reasons discussed in the next section). The second is to avoid interference with the other opamps also connected to the thermoelectric device, which we observed early on in the circuit design. At high frequencies, the real part of the Thevanin equivalent impedance at the non-inverting input of the opamp comes primarily from R_s, so R_g contributes a negligible amount to the noise. The equivalent impedance of our source, shown in

Fig. 4.2 in the red box, can be written in complex form as

$$Z_s = \frac{1}{\frac{1}{\mathrm{R}_3} + \frac{1}{-\frac{i}{2\pi \mathrm{C}_1 f} + \frac{1}{\frac{1}{\mathrm{R}_g} + 2i\pi \mathrm{C}_2 f} + \mathrm{R}_s}}, \tag{4.2}$$

where f is frequency in Hertz and the resistance and capacitance values are defined in Fig. 4.2. In our final design, $\mathrm{R_g}$ = 10 MΩ, $\mathrm{C_2}$ = 100 nF, $\mathrm{C_1}$ = 1 nF, $\mathrm{R_3}$ = 1 MΩ, $\mathrm{R_1}$ = 33.2 Ω, $\mathrm{R_2}$ = 300 Ω, and $\mathrm{R_s}$ varies around 1-5 kΩ.

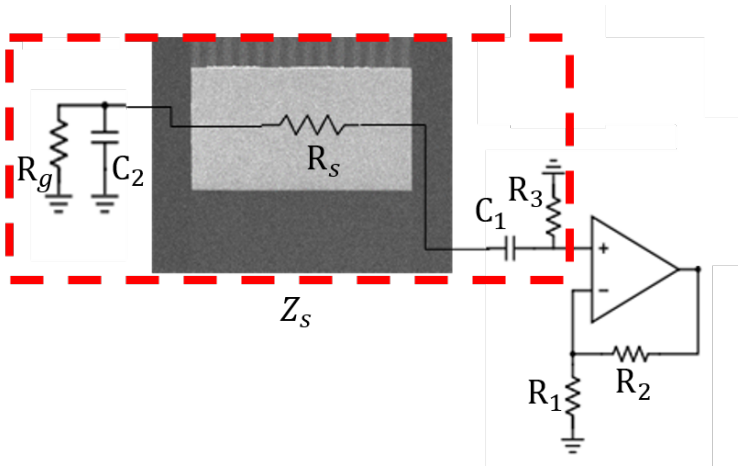

Figure 4.2: First stage of the circuit. R

A plot of the real part and the magnitude of Z_s is shown in Fig. 4.2, for $\mathrm{R_s}$ = 3.3 kΩ. At low frequencies, $\mathrm{C_1}$ acts like an effective open circuit, so the only current path available is through $\mathrm{R_3}$, which is 1 MΩ. At high frequencies, past the high-pass filter cutoff, the capacitors act like shorts so $\mathrm{R_s}$ is the primary source of impedance. Thus, past approximately 30 kHz, all of the Johnson noise will be from our thermoelectric structure, and not the circuit components.

We need to extract both the high frequency data for temperature measurements and the low frequency data for thermoelectric voltage measurements. This circuit with the given values allows us to get information for both. Data analysis will be discussed in detail in the next section, but for a 50 Hz square wave with an amplitude (1/2 V_{pp}) of 1 V, for a source resistor of 3300 Ω, we expect the signal at the input of the first opamp to be that shown in Figure 4.4.

Returning to the comparison between various opamp types, Fig. 4.5 compares the AD797 (top) with the OPA827 (bottom) for $\mathrm{R_s}$ = 3300 Ω. In these plots and the

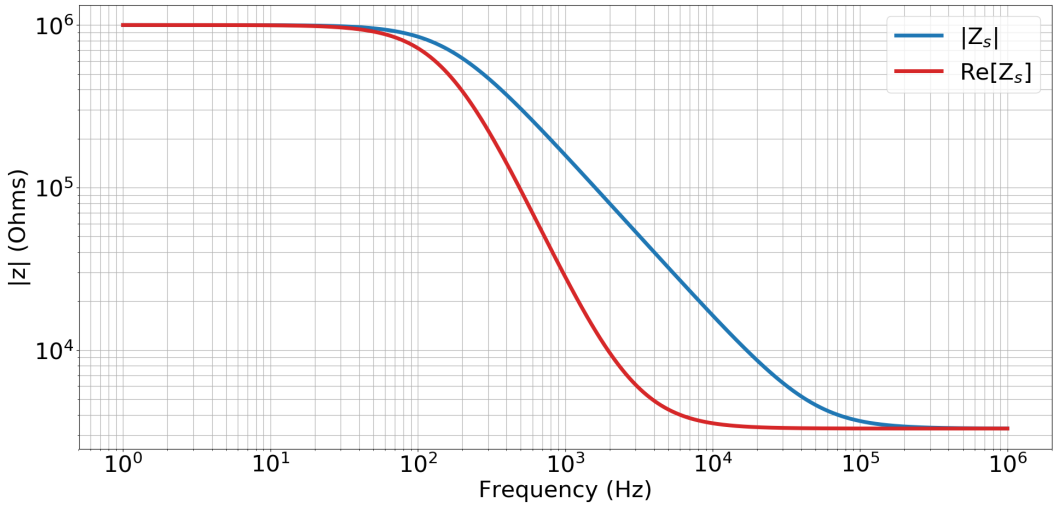

Figure 4.3: Magnitude of (blue) and real part of (red) the impedance at the non-inverting input of the 1st stage of the opamp shown in Fig. 4.2.

following plots, we ignore any amplifier gain rolloff at higher frequencies (at 10x this will be fairly small for the two studied opamps), 1/f noise (should be negligible at the high frequencies we are concerned with) and also the noise of the DAQ. The DAQ noise we observed was generally on the order of 10's of nV/$\sqrt{\text{Hz}}$, so a 10x gain would not generally be sufficient to overcome this noise source. If gain is raised to 100x, the amplifier rolloff at higher frequencies becomes significant for these two opamps. Therefore, the best option is to use a 2-stage opamp system, both with 10x gain, or whatever gain allows sufficient bandwidth for the thermometry measurements. The OPA827 has higher voltage noise and lower current noise, making it a better first stage amplifier for our system, while the AD797 has low voltage noise and higher current noise, making it a better second stage opamp for our system. We measure the noise generally above 100 kHz in our circuit, well past where the capacitors contribute to the impedance of the source. Note that the AD797 has not been tested for stability using the given gain resistors, though the data sheet does indicate it should be stable. From experience working with this opamp, pin 8, the decompensation and distortion neutralization pin, can be critical for performance. These plots represent theory calculated using values from the data sheet, not measurement, though the OPA827 is characterized further in this section experimentally. In general, commercial opamps are used for signal measurements, not noise measurements. Typically, filters are used to cut out noise in the frequencies far from the signal of interest, so much effort is not spent on creating an opamp of ideal noise characteristics at a large frequency band. We have

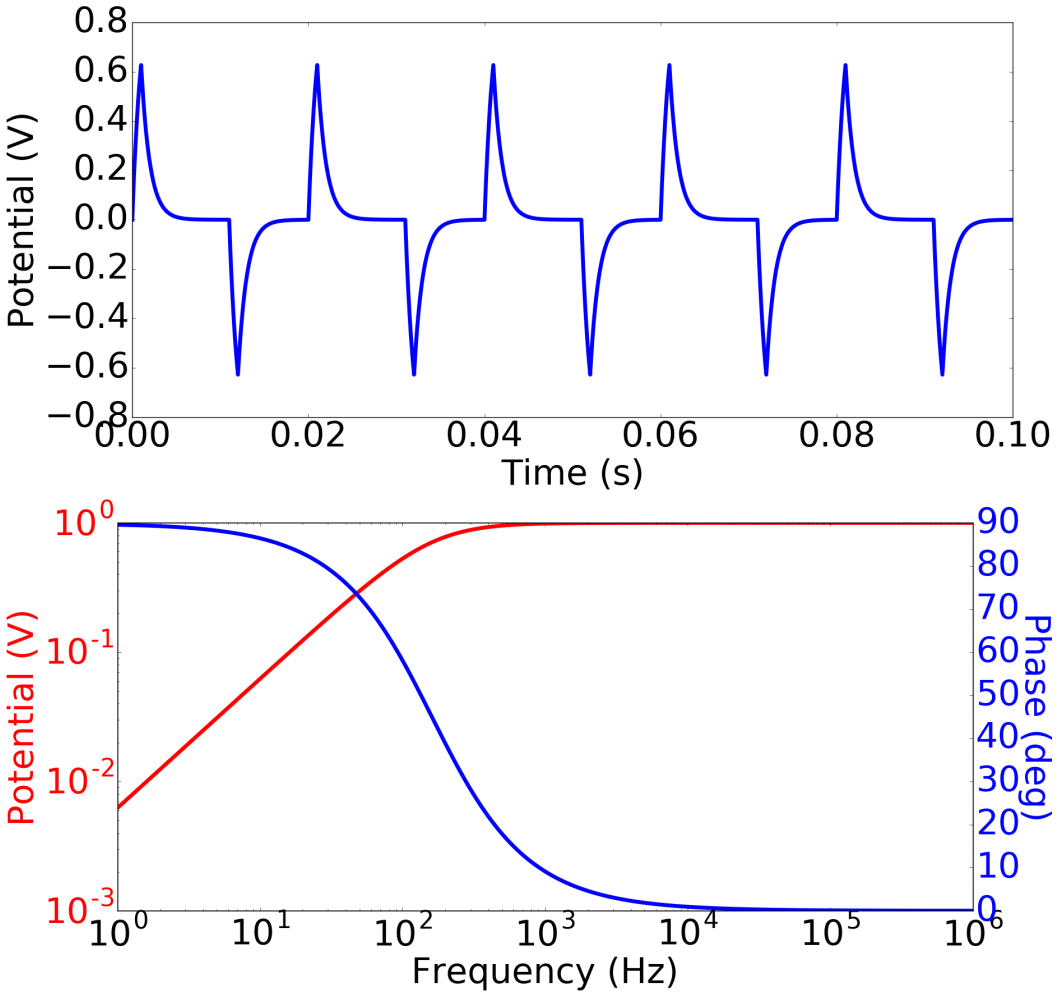

Figure 4.4: Spice simulated voltage and phase versus time (top) and frequency (bottom) seen at the input of the opamp. The signal is a square wave at 55 Hz with 1 V amplitude generated across the source resistor, which is 3300 Ω.

found that opamps with external compensation pins are more difficult to stabilize across a broad frequency range and get close to ideal noise conditions. Opamps, like the OPA827, require minimal external compensation tweaking as they have no compensation pins to tweak, and are therefore self-compensated and behave generally well. Thus, we used the OPA827 for both the first and second stage opamp in our circuit, as the noise variation from opamp to opamp seemed to be be much smaller than the other tested opamps, and we found it easier to work with, though the AD797 would be the better choice if an additional iteration of the circuit were to be made.

One important point in Fig. 4.5 is that while the current noise power is a negligible

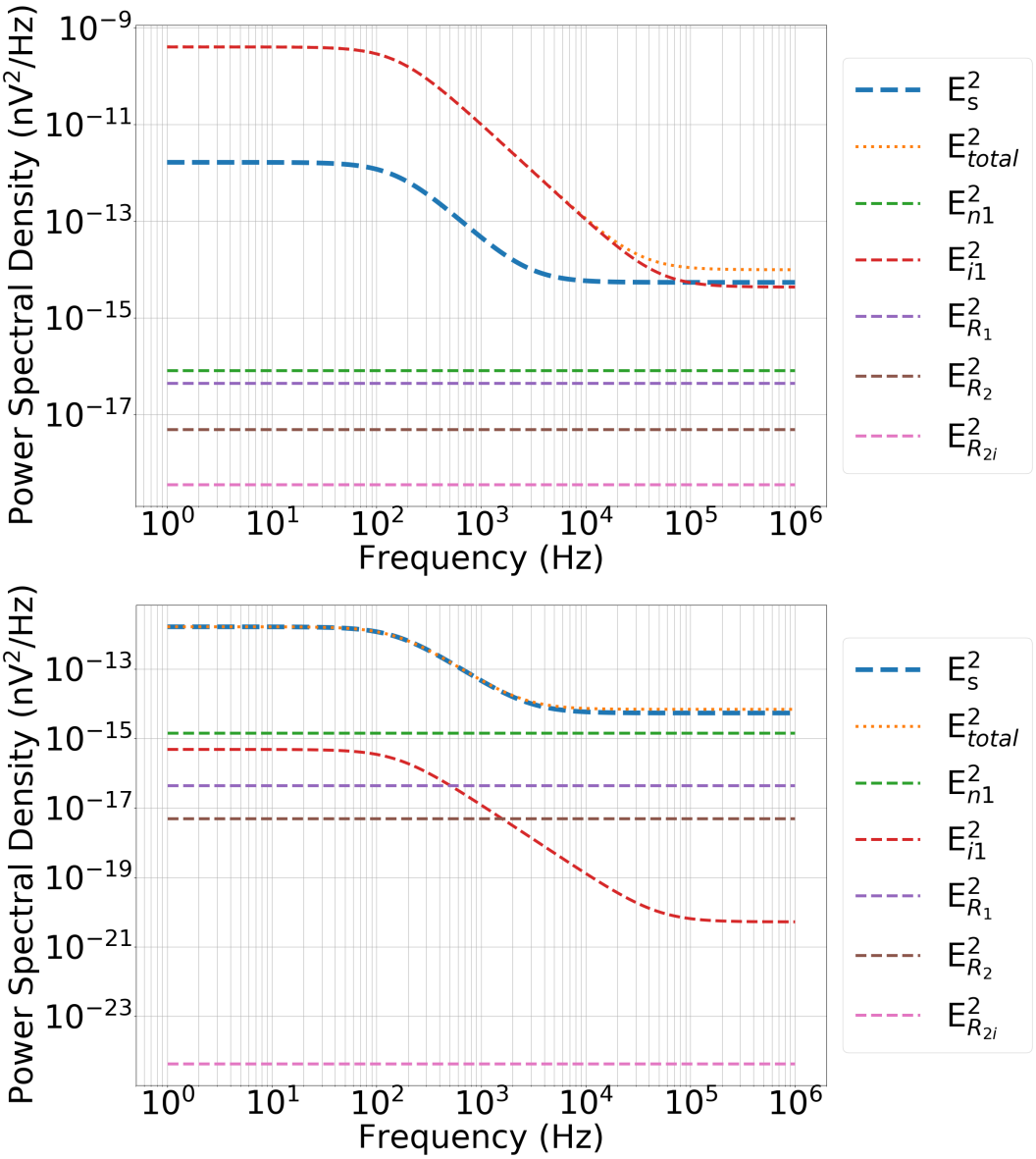

Figure 4.5: Comparison between the power spectral density (PSD) of AD797 (top) and OPA827 (bottom) for an R_s of 3300 Ω. E_s^2 is the noise power density from R_s, E_{total}^2 is the sum of all noise power sources, E_{n1}^2 is the voltage noise power of the opamp, E_{i1}^2 is the current noise power of the opamp acting on $|Z_s|$, $E_{R_1}^2$ is the voltage noise power of R_1, $E_{R_2}^2$ is the voltage noise power of R_2, and $E_{R_{2i}}^2$ is the current noise power of the opamp current noise acting on R_2 (ignoring any stray capacitances that may contribute).

contribution to the total noise for the OPA827, it is nearly equal to the source voltage noise in the AD797. Therefore, the OPA827 is a much better choice for this resistance, as the source resistance makes up the largest component of the total

noise. This effect is exaggerated further if the source resistance is increased to 10 kΩ, as shown in Fig. 4.6. In this case, for the AD797 (top of Fig. 4.6, the opamp current noise power is clearly the dominant noise source, compared to the OPA827 (bottom of Fig. 4.6), where the current noise power is orders of magnitude lower than the sample noise power.

Conversely, if our source resistance is smaller, the AD797 is a better choice. Figure 4.7 shows the power spectral density contributions for a source resistance of 500 Ω as measured by the AD797 (top) and the OPA827 (bottom). The opamp voltage noise power is higher than the noise power of the source resistance, so this opamp would not be suitable for this noise measurement. The AD797, despite having 3 orders of magnitude higher opamp current noise than the OPA827, has low enough Johnson current noise above about 200 kHz to render this opamp usable for this noise measurement application. Furthermore, increasing the values of the coupling capacitors could reduce the minimum measurable frequency in this application with the AD797.

In JFET amplifiers (JFET transistors are found in most low-noise amplifiers sold today), the current noise doubles for every 10-20 K temperature rise in the circuit (see any JFET input opamp Data Sheet plot of input bias current versus T, such as the OPA827). Therefore, it is important to first thermally manage the circuit, including using the lowest possible power supply to the opamp, as much power dissipation happens at or near the opamp. An easier and likely more effective solution, as heat-sinking opamps can be difficult or not possible in small spaces, one should choose a source resistance and opamp for which if the current noise doubles or triples, the output noise will be unaffected. Figure 4.8 shows the voltage noise produced by varying current noise for a number of different source resistance values (noise is relative to output, and for a 2-stage circuit, which will be discussed shortly). The OPA827, with 2.2 fA/$\sqrt{\text{Hz}}$ current noise, would have a negligible contribution to the overall noise of the circuit if multiplied by a factor of 10, which makes it ideal for use in applications where the environment temperature cannot be carefully controlled.

Figures 4.9,4.10 show how the output noise changes as a function of current noise and source resistor.

One important note is that instrumentation amplifiers must be very carefully selected if used in noise measurements. Most have very high current noise, stemming from internal input bias current compensation. While the input bias current of these inamps can be small, there are sufficient internal currents carefully balancing these

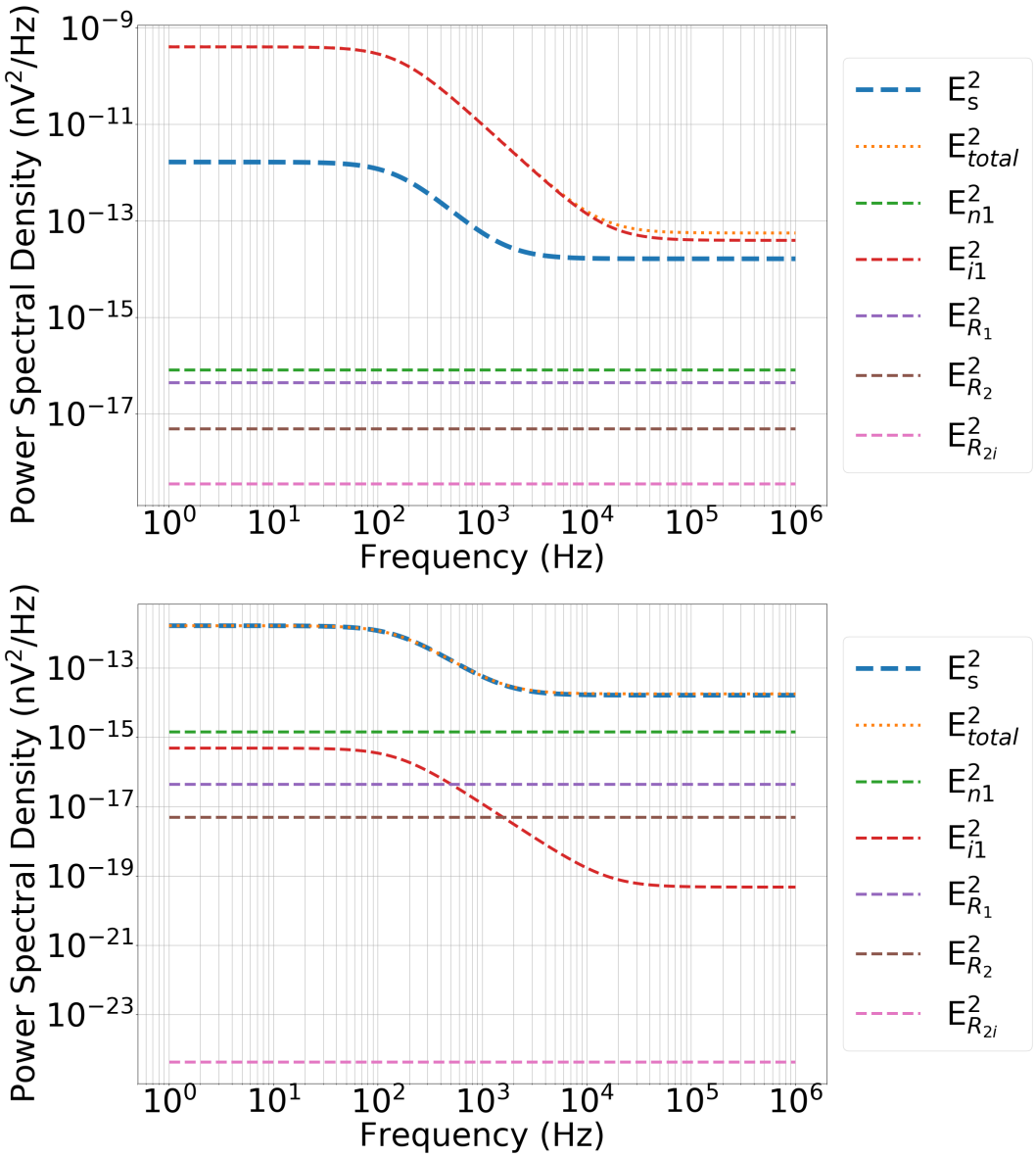

Figure 4.6: Comparison between the power spectral density (PSD) of AD797 (top) and OPA827 (bottom) for an R_s of 10 kΩ. E_s^2 is the noise power density from R_s, E_{total}^2 is the sum of all noise power sources, E_{n1}^2 is the voltage noise power of the opamp, E_{i1}^2 is the current noise power of the opamp acting on $|Z_s|$, $E_{R_1}^2$ is the voltage noise power of R_1, $E_{R_2}^2$ is the voltage noise power of R_2, and $E_{R_{2i}}^2$ is the current noise power of the opamp current noise acting on R_2 (ignoring any stray capacitances that may contribute).

.

output currents to minimize DC offset, that generate excessive current noise.

The source resistances across our thermoelectric devices are controllable via the

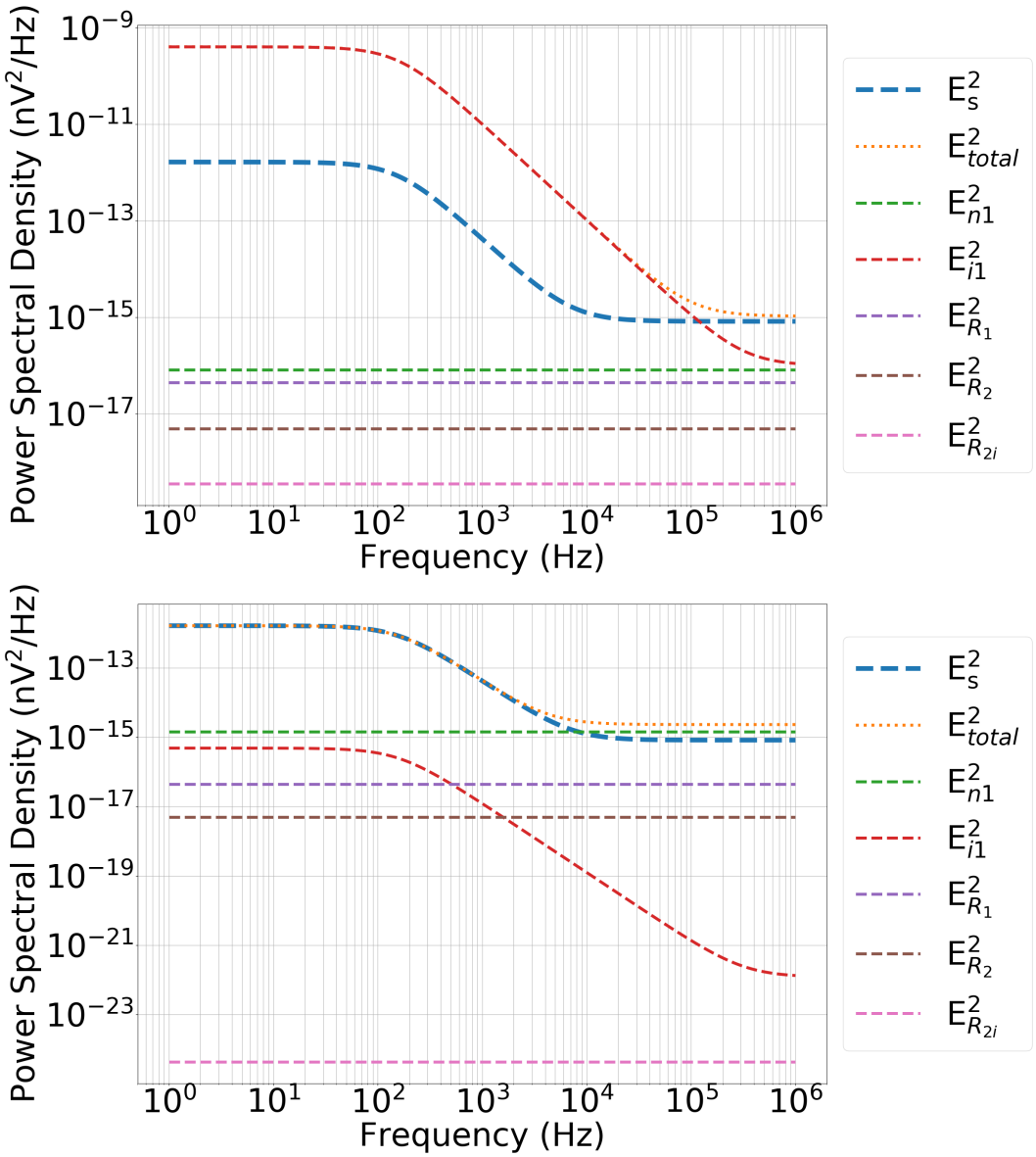

Figure 4.7: Comparison between the power spectral density (PSD) of AD797 (top) and OPA827 (bottom) for an R_s of 500 Ω. E_s^2 is the noise power density from R_s, E_{total}^2 is the sum of all noise power sources, E_{n1}^2 is the voltage noise power of the opamp, E_{i1}^2 is the current noise power of the opamp acting on $|Z_s|$, $E_{R_1}^2$ is the voltage noise power of R_1, $E_{R_2}^2$ is the voltage noise power of R_2, and $E_{R_{2i}}^2$ is the current noise power of the opamp current noise acting on R_2 (ignoring any stray capacitances that may contribute).

material stoichiometry and thickness. While it would be most desirable to use a low source resistance material, such as a metal like chromium, which is on the lower end of thermal conductivity for metals and has a decent Seebeck coefficient,

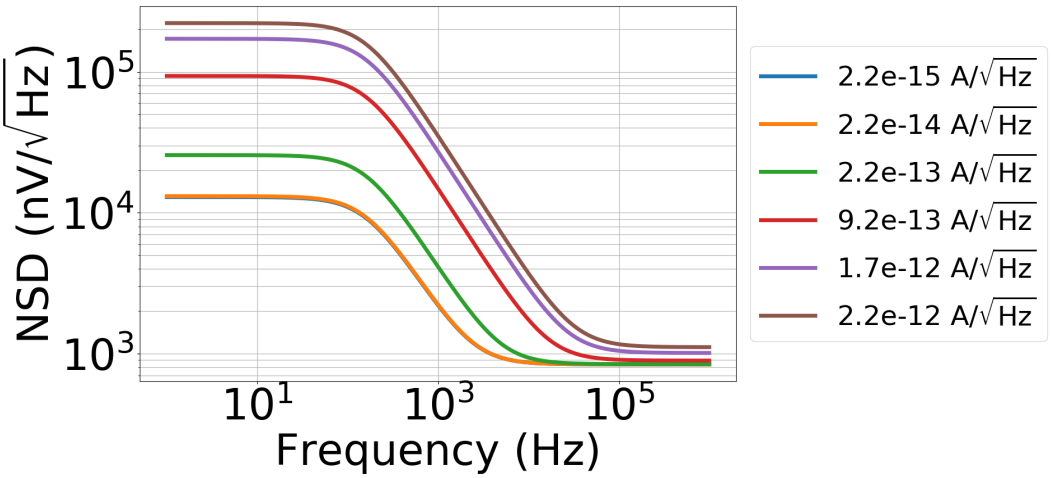

Figure 4.8: Variation of output noise spectral density with current noise on a 2-stage OPA827 opamp circuit for R_s = 3300Ω.

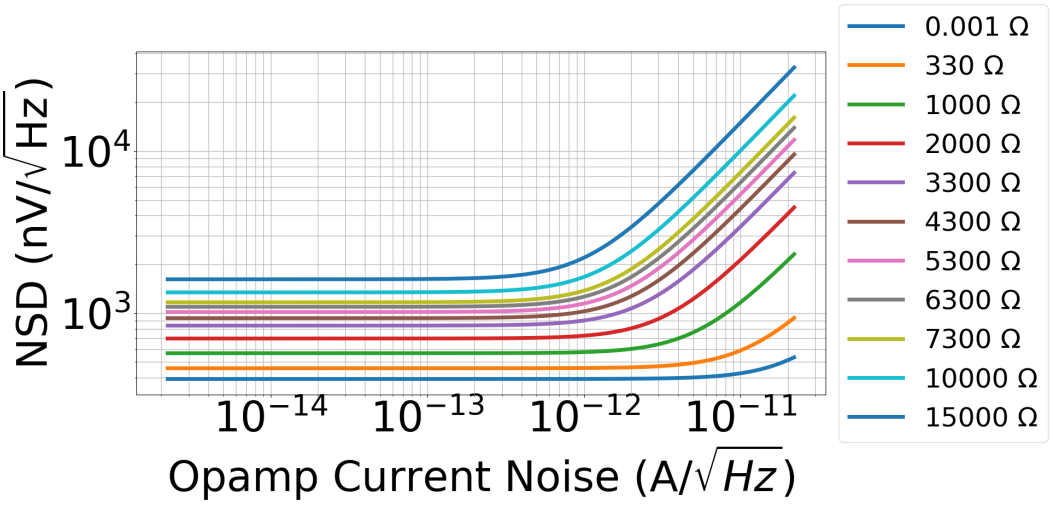

Figure 4.9: Variation of output noise spectral density with source resistance and current noise on a OPA827 2-stage opamp circuit.

these materials have drawbacks. Because the size of the device is so small, the resistance of the gold leads to the device (on the order of 50-100 Ω depending on lead geometry) will dominated the resistance of the device, which will be less than 10 Ω for a metal. Having a resistance that is too high, such as 100 kΩ, is also undesirable, due to the stray capacitances on the PCB board, as we have discovered. About 5 pF is a typical stray capacitance value for our signal leads on our PCB board without excess engineering to account for it, which when combined with a large resistance, creates a low-pass filter, and can cut-off noise signal in the higher

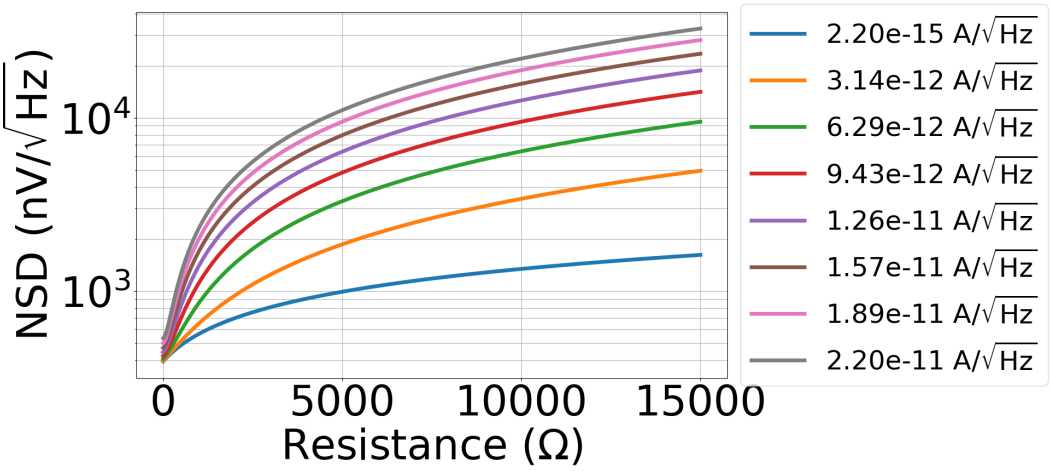

Figure 4.10: Variation of output noise spectral density with source resistance and current noise on a OPA827 2-stage opamp circuit.

frequencies. By controlling the stoichiometry and thicknesses of our materials, we aimed for a resistance of around 4 kΩ, which yielded a theoretical Johnson noise above the opamp voltage noise of the OPA827, but was not a large enough resistance to be affected at high frequencies by stray capacitances.

As stated previously, a 2-stage opamp system is desirable in our case, as gain higher than 10x is needed to overcome the DAQ noise, but the low-noise opamps available to us do not have the gain-bandwidth product necessary to have greater than 10x gain without significant amplifier gain rolloff at higher frequencies. Figure 4.11 shows our 2-stage amplifier circuit. It is critical for the first stage amplifier to have low-noise characteristics, but the second stage amplifier has more relaxed noise requirements, as its contribution to the output noise will be less by a factor of the first stage gain than the first stage amplifier noise. Additionally, while a non-negligible source resistance generates extra noise from the output current noise of the first stage amplifier, the output impedance of most amplifiers is very low, so the current noise of the second stage opamp will not generate much extra noise at the output. The best amplifier to use at the second stage would be an amplifier with a very large gain bandwidth product, such as the AD829, with a 600 - 750 MHz gain bandwidth product. This would allow for a very large gain on the second stage. In our experience, the AD829 was found to have high variation in noise between different amplifiers, and was tricky to work with initially. For this reason, we used the OPA827 as the second stage amplifier, although in a second generation device, optimizing the compensation and layout of the OPA827 would be worth the time

spent. In Figure 4.11, $R_1 = R_4$ and $R_2 = R_5$, which are given above.

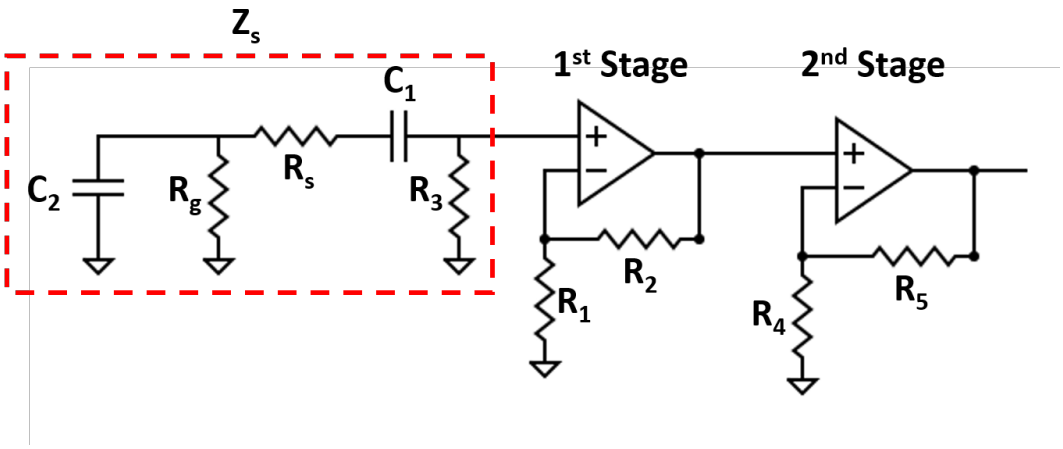

Figure 4.11: 2-stage circuit.

Figure 4.12 shows the power spectral density referred to the output of a 2-stage opamp circuit shown in Figure 4.11, with the values of the circuit elements defined previously, and $R_1 = R_4$ and $R_2 = R_5$, giving each stage approximately 10x gain each. The total output noise of this circuit is given by

$$\begin{aligned} E_0^2 = &\left[4k_bT_sRe[Z_s]\left(\frac{R_2}{R_1}+1\right)^2 + (i_{n1}|Z_s|)^2\left(\frac{R_2}{R_1}+1\right)^2\right. \\ &\left. + e_{n1}^2\left(\frac{R_2}{R_1}+1\right)^2 + 4k_bR_2T_{R_2} + (i_{n1}|R_2|)^2 + 4k_bR_1T_{R_1}\left(\frac{R_2}{R_1}\right)^2\right]\left(\frac{R_5}{R_4}+1\right)^2 \\ &+ 4k_bT_1Re[Z_1]\left(\frac{R_5}{R_4}+1\right)^2 + (i_{n2}|Z_1|)^2\left(\frac{R_5}{R_4}+1\right)^2 \\ &+ e_{n2}^2\left(\frac{R_5}{R_4}+1\right)^2 + 4k_bR_2T_{R_5} + (i_{n2}|R_5|)^2 + 4k_bR_1T_{R_4}\left(\frac{R_5}{R_4}\right)^2 + NSD_{DAQ}^2, \end{aligned} \tag{4.3}$$

where Z_1 is the output impedance of the first stage opamp, i_{nx} and e_{nx} refer to the current noise and voltage noise, respectively, for the first stage opamp ($x = 1$) and second stage opamp ($x = 2$), and NSD_{DAQ} is the Johnson noise of the DAQ input.

It is important to keep in mind that while it is useful to examine each noise thermometry circuit on its own, the entire circuit is shown in Figure 4.13. The resistor between each noise thermometry stage, we assume to be much larger than the R_s for each 2-stage opamp circuit, so the circuit should not interfere.

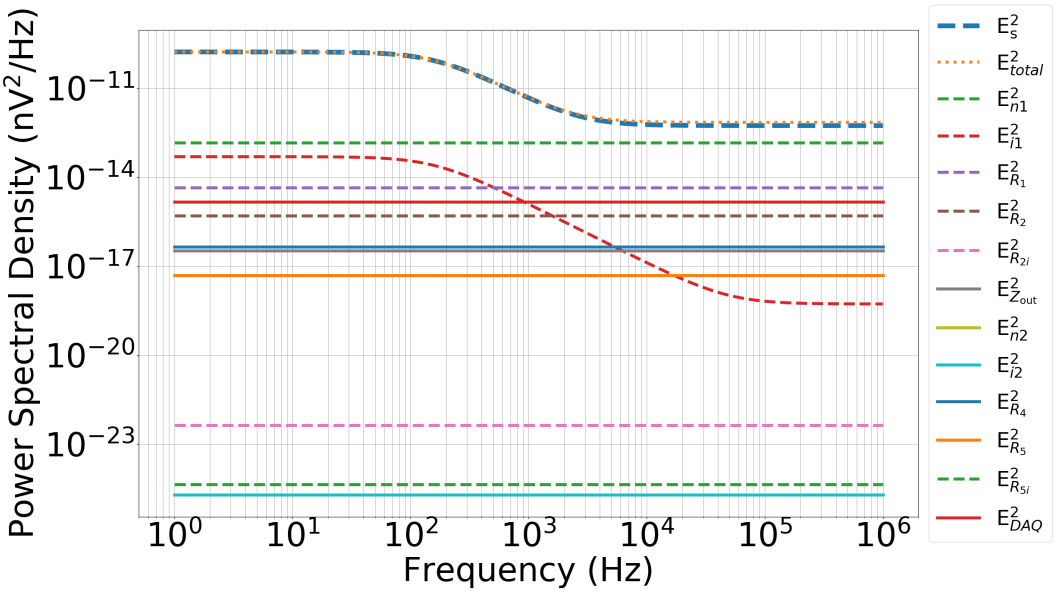

Figure 4.12: Power spectral density versus frequency for each individual contribution to the output noise. Here, R_s is 3.3 kΩ. E_s^2 is the noise power density from R_s, E_{total}^2 is the sum of all noise power sources, E_{n1}^2 is the voltage noise power of the first stage opamp, E_{i1}^2 is the current noise power of the first stage opamp acting on $|Z_s|$, $E_{R_1}^2$ is the voltage noise power of R_1, $E_{R_2}^2$ is the voltage noise power of R_2, $E_{R_{2i}}^2$ is the current noise power of the first stage opamp current noise acting on R_2 (ignoring any stray capacitances that may contribute), $E_{Z_{out}}^2$ is the noise power density from the output impedance of the OPA827 (20 Ω), E_{n2}^2 is the voltage noise power of the second stage opamp, E_{i2}^2 is the current noise power of the second stage opamp acting on $|Z_{out}|$ (the output impendance of the first stage opamp), $E_{R_4}^2$ is the voltage noise power of R_4, $E_{R_5}^2$ is the voltage noise power of R_5, $E_{R_{5i}}^2$ is the current noise power of the opamp current noise acting on R_5 (ignoring any stray capacitances that may contribute), and E_{DAQ}^2 is the noise from the DAQ we used in our measurements (National Instruments USB 6366).

A note on capacitors: the knowledge in the literature or internet about noise levels of various capacitor types is varied and sometimes contradictory. We observed no noticeable difference when using mica versus ceramic (low and high quality) versus polypropylene film in our circuit, at least at the higher frequencies we were interested in. Presence of excess flux on the circuit board also did not contribute noticeably to noise levels. Shielding, however, is extremely important to avoid high frequency signals from the environment. A grounded metal box surrounding the circuit sufficed.

Figure 4.14 shows the raw voltage data collected from the DAQ under chopped illumination of a thermoelectric sample. Note its similarity in shape to Figure 4.4.

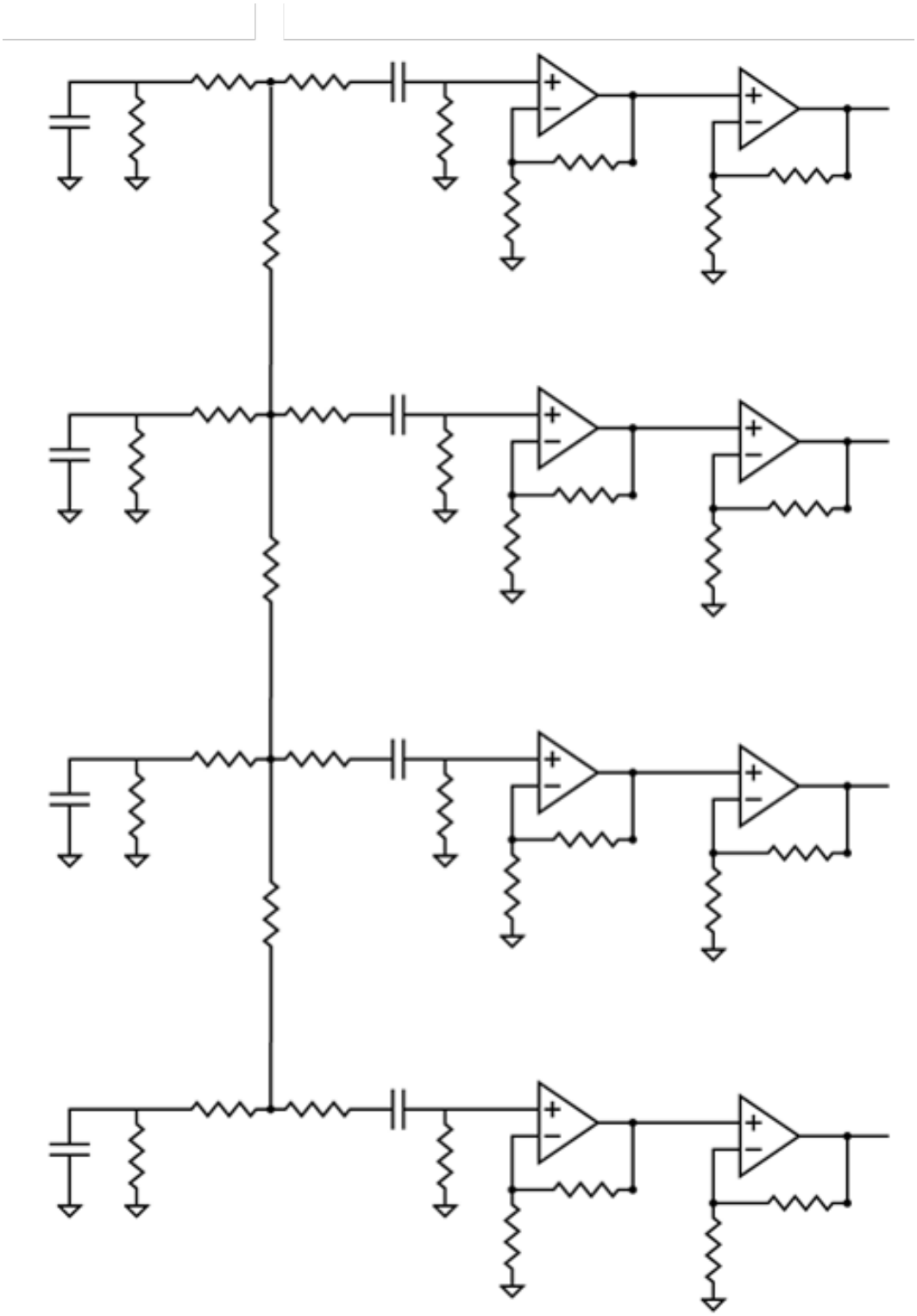

Figure 4.13: Total noise thermometry circuit.

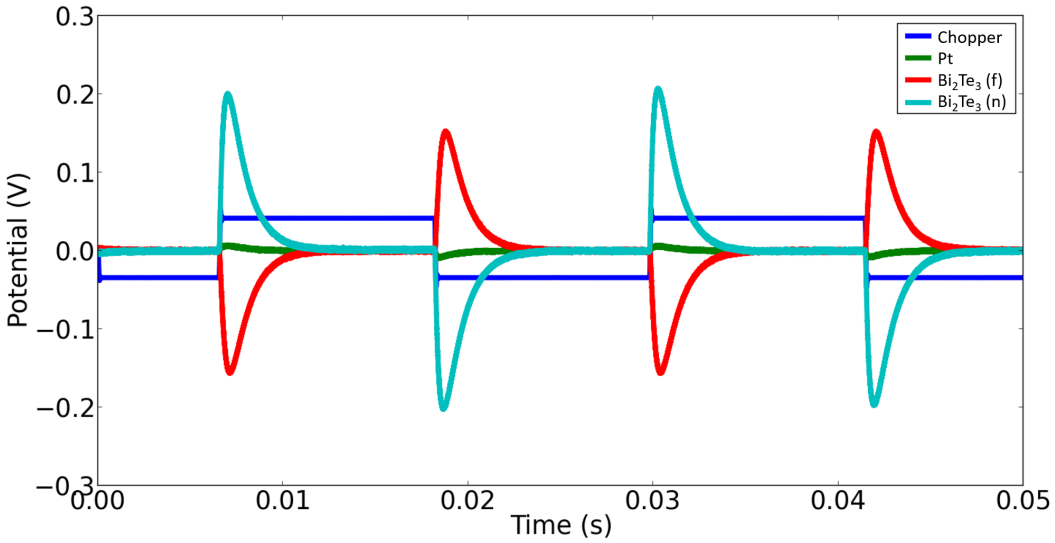

Figure 4.14: Measurement of thermoelectric sample under 639 nm, 43 Hz chopped illumination with 590 μW of absorbed power. The thermoelectric voltage of the bismuth telluride close to the wires is shown in turquoise, the thermoelectric voltage of the bismuth telluride far from the wires is shown in red, the thermoelectric voltage of the platinum close to the wires is shown in green, and the chopper voltage signal (smoothed digitally with a low pass filter) is shown in blue.

Figure 4.15 shows data collected from our circuit, plotted with two different axes. There is a small oscillation at lower frequencies, but as we are measuring the frequencies between 100,000 and 900,000 Hz, this is tolerable.

Measuring resistance changes across leads

In order to account for noise changes due to resistance changes, we precisely measured resistance in our samples by inserting a small current at a certain frequency and monitoring the resulting change in magnitude of the voltage it produced during the course of noise measurements. Thermoelectric voltage measurements were performed below 100 Hz and noise measurements were performed above 100 kHz, so the resistance measurements were performed between 18 - 22 kHz, as at these frequencies the source resistance noise dominates (Figure 4.12). Two SRS 830 lock-in amplifiers were used as signal generators to produce a small current (nanoamps) at the desired frequencies. The insertion point of this signal is shown in Figure 4.16, along with the resistor and capacitor values in the bandpass filter. In designing the filter, an emphasis was put on decreasing the lock-in noise contribution in the 100 kHz and higher frequency regime, which meant that the bandpass filter is not centered on the resistance measurement frequencies of interest, but allows through

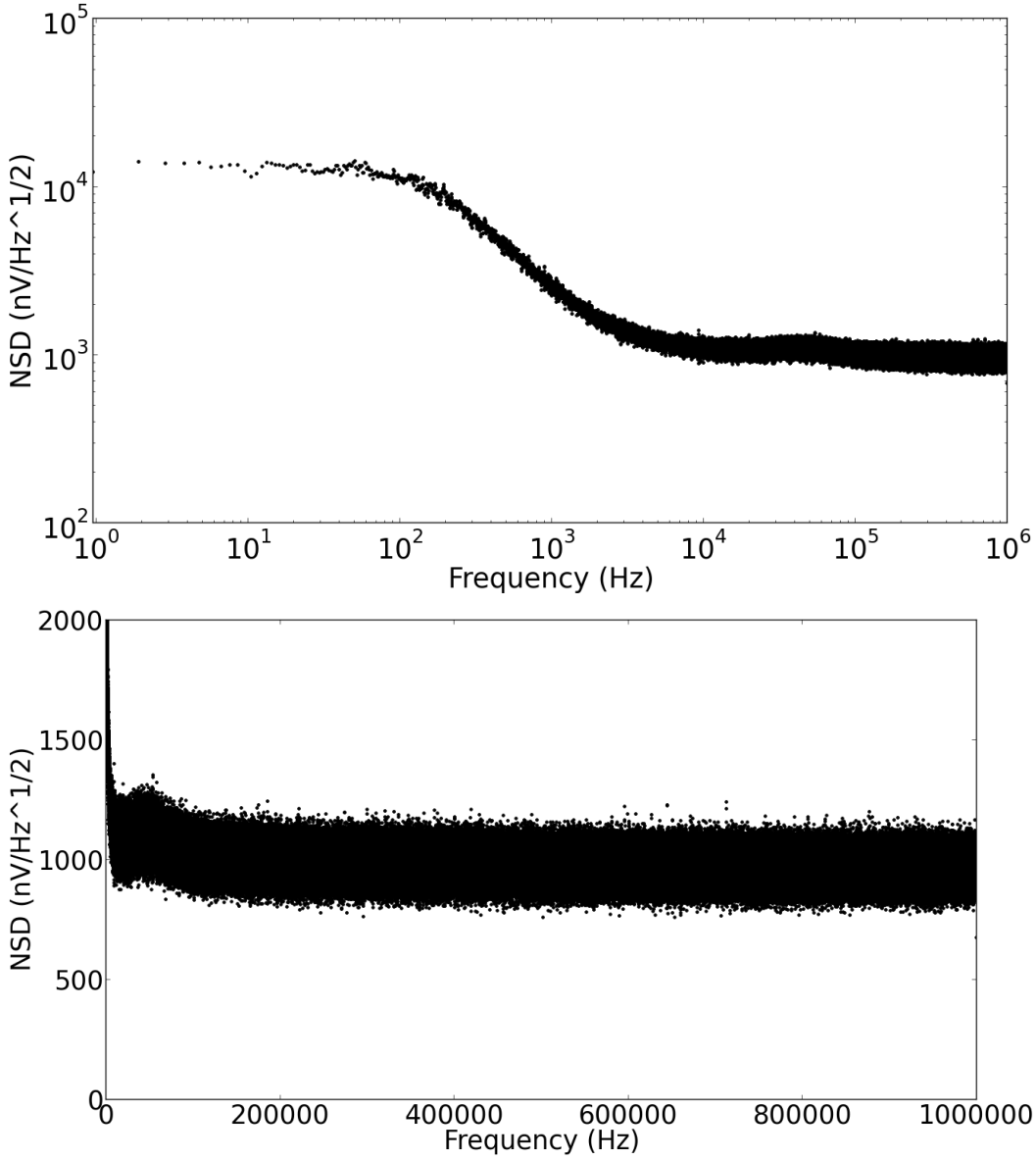

Figure 4.15: Measurement from DAQ, including gain and circuit noise (plotted with different axis), with $R_s = 3.3k\Omega$ metal film resistor.

enough signal for our purposes. This filter could be optimized further.

Figure 4.17 shows the resulting NSD spectrum on our thermoelectric sample with a resistance measurement signal. The change in magnitude of the resistance measurement with power is amplified by the opamps and analyzed well the noise data as described in a later chapter.

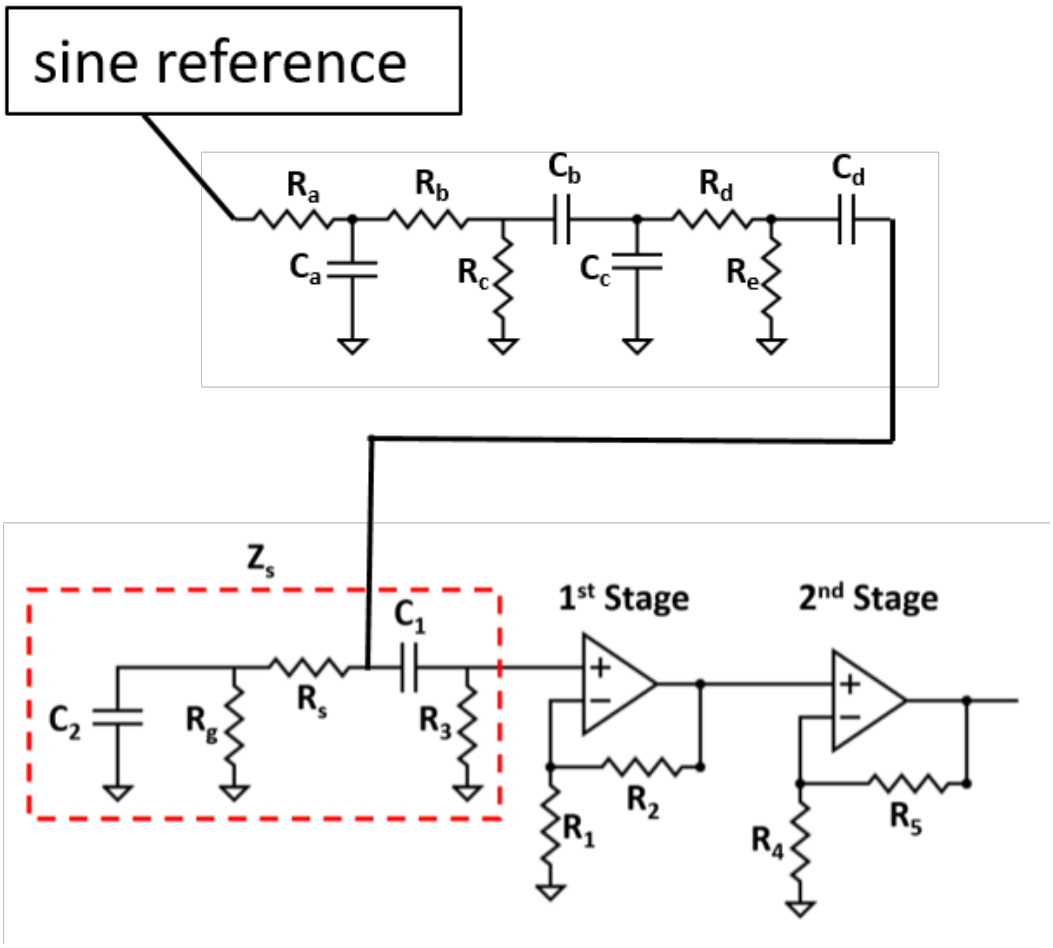

Figure 4.16: Second-order bandpass filter for injecting resistance-measuring current while filtering high and low frequency noise produced by the signal generator (the sine reference output of a SRS 830 lock-in amplifier, in this case). Here, R_a is 1 kΩ, and R_b, R_c, R_d, and R_e are 1 MΩ each. C_a, C_b, C_c, and C_d are 100 nF, 1 nF, 0.1 nF, and 1 nF, respectively. The other resistor and capacitor values are given previously in the text.

Thermal design of board

As discussed earlier, temperature increases in opamps can increase current noise significantly above datasheet values, which is undesirable for accurate noise thermometry measurements. We discovered this using an early version of our circuit which used AD8421 instrumentation amplifiers and a ±15V power supply, a large amount of heat was being dissapated in the inamps, due to the large power supply and small signal outputs. Figure 4.18 shows an image of our circuit next to a IR camera image, illustrating the temperature rise of about 10 K in the inamps after only 5 minutes of operation. At this point, we were using the NI-USB 4431 with 100 kHz

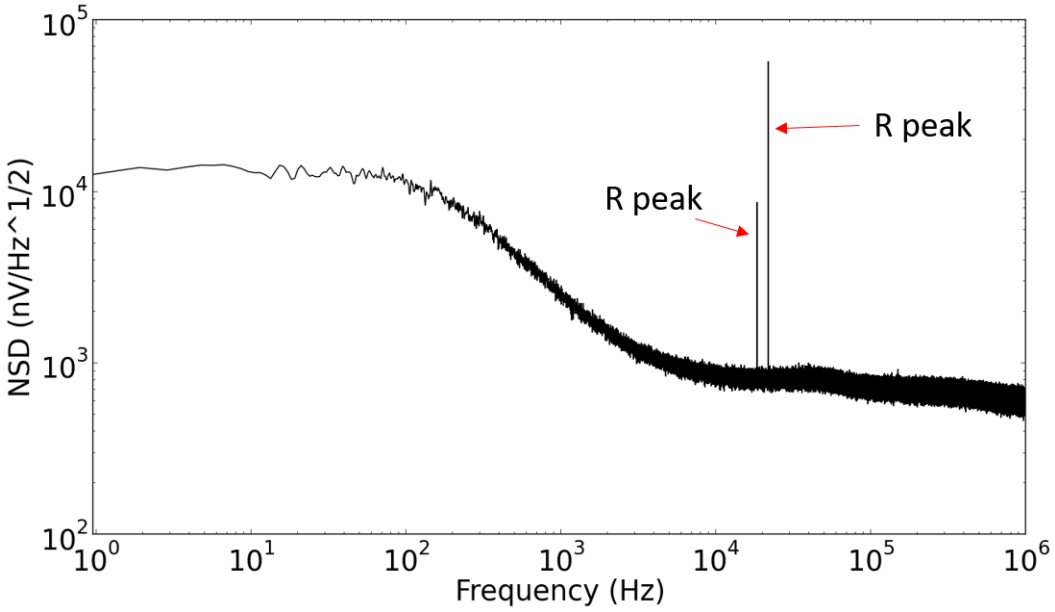

Figure 4.17: NSD spectrum with resistance signal.

data collection rate, so measurements took hours, which would have significantly heated the circuit board and the sample, and also increase the inamp current noise output.

One data set taken with this circuit is shown in Figure 4.19. Laser power was swept from high to low power over the course of several hours, and the overall drift in temperature extracted from the noise was about 60 K, which we attribute to addition of current noise over time due to inamp heating. We later discovered, as we will show, that the resistance changes may also have accounted for some of the drift in noise with time, however, with this circuit, we did observe increased noise over time scanning from both high to low and low to high laser powers, which brings us to conclude that the circuit heating did have a non-negligible effect on this noise rise.

A number of changes were made to correct these issues. First, as the signals we were looking at were small, we replaced the ±15 V power supply with ±5 V power supplies, the minimum value for most amplifiers. This would decrease the power dissipated across the amplifier. Because we lowered the power supply rails, we had to decrease the AC coupling resistor in front of the inamp from 10 MΩ to 1 MΩ, as the input bias current offsets sometimes gave a DC input voltage that exceeded the rails when amplified. This had negligible effect on noise. As we were using the lowest current noise inamp available (200 pA/$\sqrt{\text{Hz}}$, we switched to the opamp configuration shown in the preceeding section, as there existed opamps

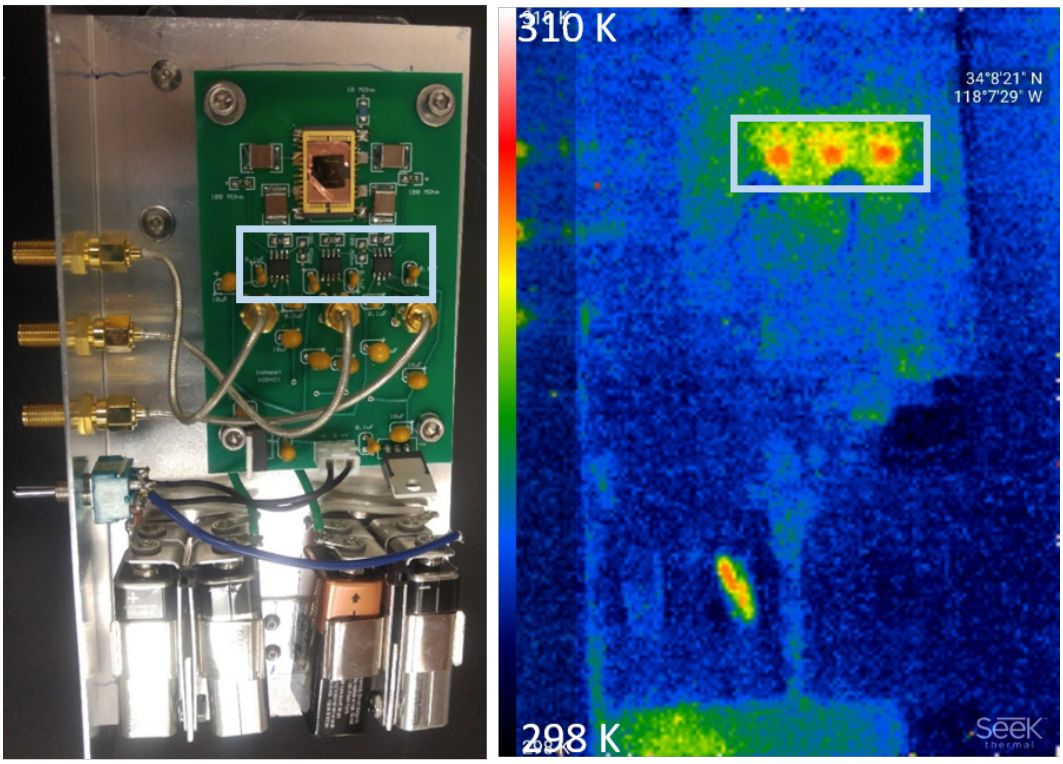

Figure 4.18: Heat generation in inamps in early circuit after 5 minutes of operation. Hot spots near the bottom and left side of the image were due to reflections off metal surfaces, and did not correspond to actual temperatures (these "hot spots" disappeared when pictured from different angles, while the inamp hot spots remained).

with current noise so low that a 50 K increase in circuit board temperature would still have no effect on the circuit board. Finally, we reduced the 9V batteries from 4 to 2 to decrease power dissipation across the voltage regulators, and moved the batteries and voltage regulators to a separate box (Figure 4.20) so as to not affect the circuit board temperature. We also installed 2 surface mount RTDs (resistive thermometers) to monitor the circuit temperature, and connected these to a DAQ (OM-USB-TEMP-AI from Omega) to collect circuit board temperature data during noise collection. We installed DC fans to further cool the circuit, but these added peaks to the noise spectra, so we eliminated them. We also redesigned the circuit to keep the opamps as far as possible from the sample. All of these changes sufficed to fix the heating problems.

The final circuit is shown in Figure 4.21.

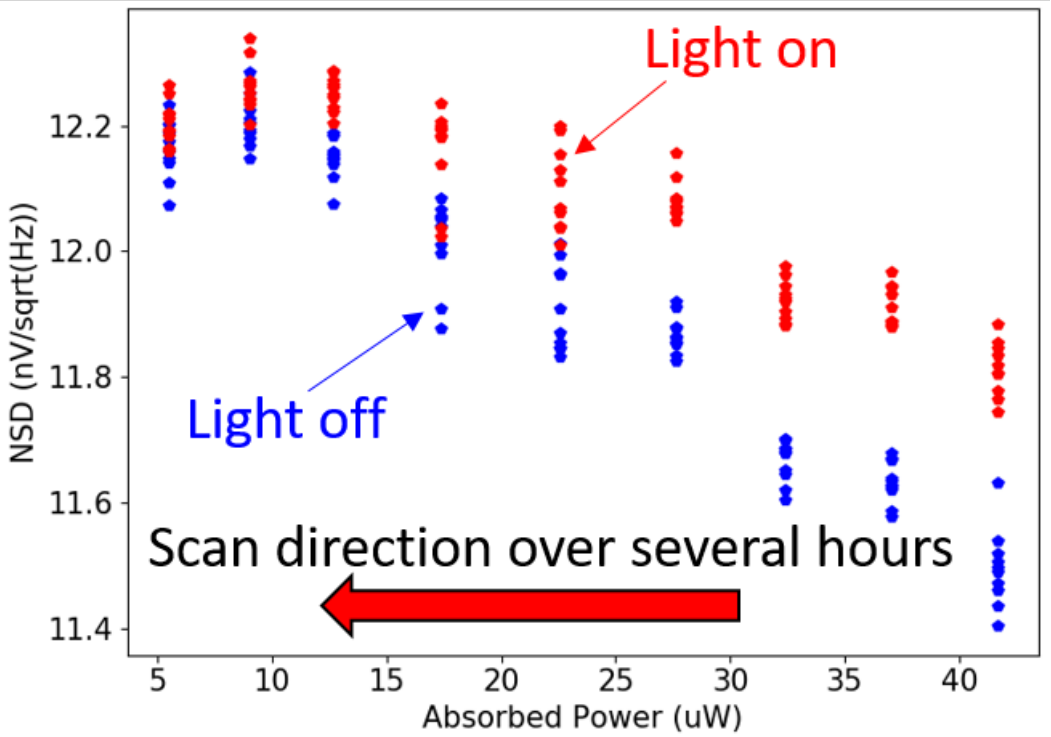

Figure 4.19: NSD as a function of absorbed laser power in thermoelectric device. This data collection took several hours, and the laser power was swept from high power to low power. The approximate temperature drift in the measurement corresponds to about 60 K, which we attribute to an increase in current noise from the inamp due to heating, as dicussed in the text.

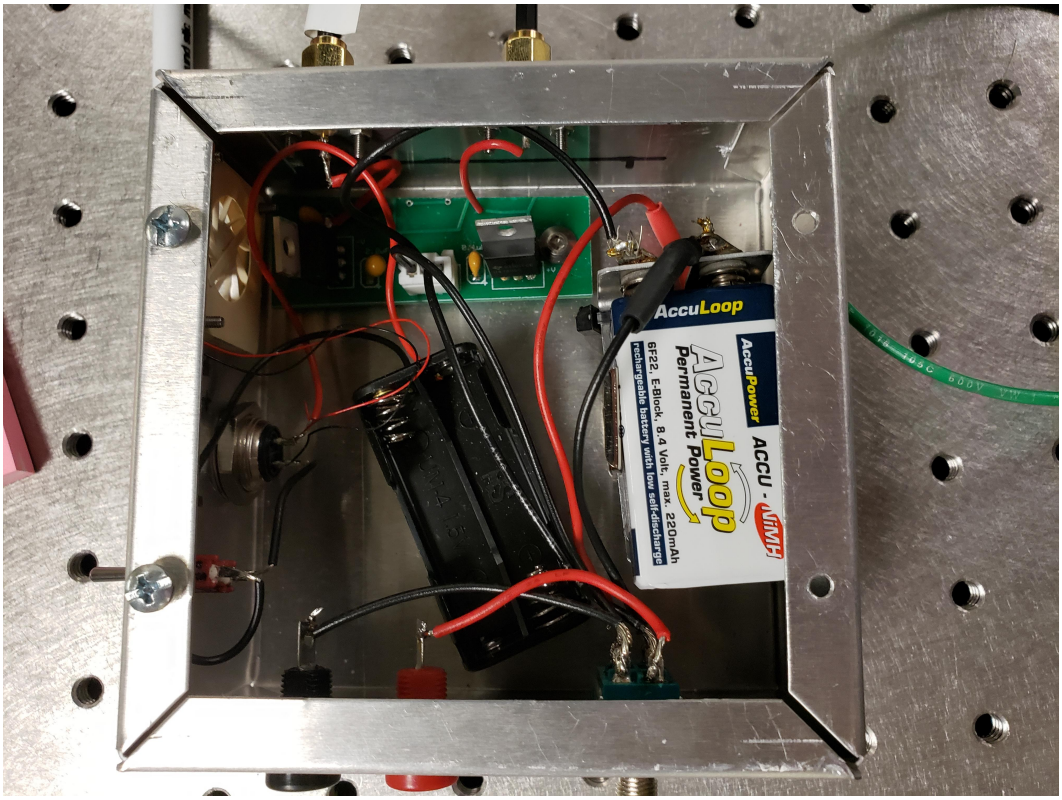

Figure 4.20: Separate box for batteries.

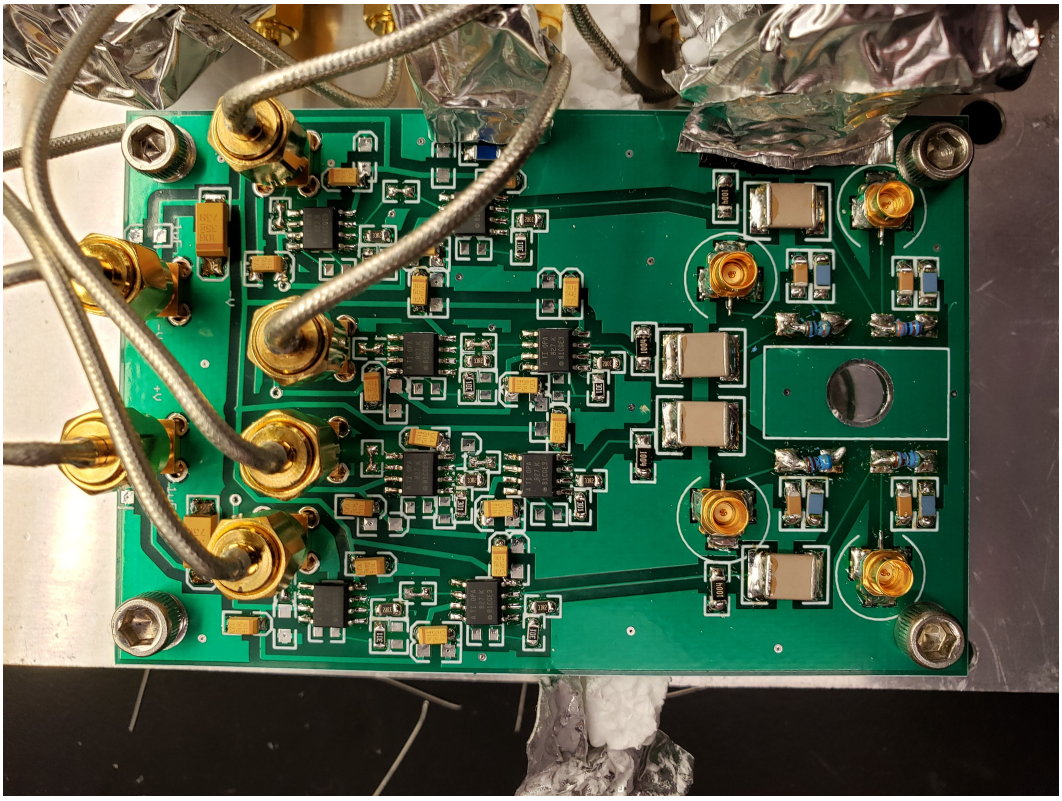

Figure 4.21: Final circuit.

Chapter 5

NOISE THERMOMETRY RESULTS

5.1 Data analysis

The data analysis process was generally split into two pieces. The first piece is determining the noise temperature under chopped illumination or without illumination, and the second piece, if under chopped illumination, was to determine the magnitude of the thermoelectric voltage produced. Both will be discussed in subsequent sections. Because of the large volume of data involved (billions of data points), C was used with the FFTW (Frigo and S. Johnson, 2005) libraries, which was several orders of magnitude faster than Python scripts (many hours with Python versus 20 minutes with C). Both C and Python scripts can be found in the appendices.

Determining noise temperature with Welch's method

To determine the temperature from our measurements, the process is as follows (the C analysis script can be found in the appendix A):

1. Read large binary file for each channel plus chopper signal.
2. Separate the data into "on" and "off" data using the chopper signal (this step is skipped in a control run with no laser signal).
3. Generate a Welch's periodogram from the "on" and "off" data separately.
4. Average together the high frequency data to get the power spectral density (PSD).
5. Take the square root of the averaged data to get the noise spectral density (NSD).

Note that NSD must be averaged in quadrature (i.e. the mean of the PSD is taken) to avoid underestimating the noise. Figure 5.1 outlines Steps 2-4.

Step 2 can be accomplished with a computer-generated chopper signal if proper synchronization between instruments and computers can be achieved, otherwise the chopper and signal will drift relative to one another. In our case, we read in the

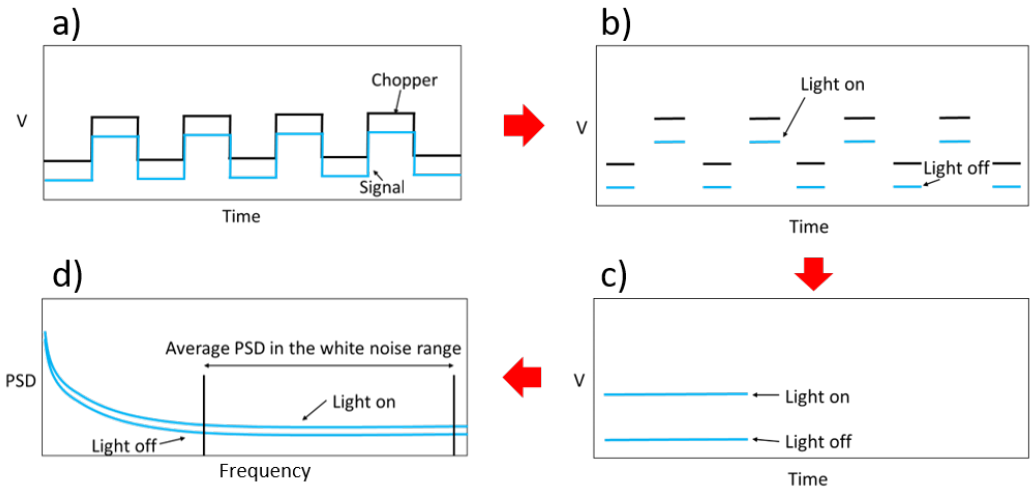

Figure 5.1: Sequence for analyzing noise data, as described in Steps 2-4 in the text. Note that the signal wave is not a square wave, as is depicted here in a) for simplicity. See Figure 4.14 for measured signal shape. b) shows the deletion of transition regions from the chopper and signal traces. c) depicts joining the "light on" and "light off" data into long data arrays. d) shows a cartoon PSD trace from the "light on" and "light off" data showing the expected increase in noise for the "light on" data.

chopper signal from the waveform generator or chopper from the DAQ and lowpass filtered the signal in order to find transition regions (the sharp on/off points in the chopper signal). The non-flat regions (when currents were still flowing) to the left and right of the sharp spike were deleted, as the Nyquist noise theorem only applies to systems in equilibrium. This non-equilibrium noise data could be interesting to examine in the future, but was not the focus of the present study. The "on" data was put into one data set, and the "off" data was put into a different data set. These were plotted to make sure they looked relatively flat to ensure the transition regions had been cut effectively.

The "on" and "off" data then went through a process to generate a Welch Periodogram (Welch, 1967). A discussion of different methods of taking periodograms can be found in (Stoica, Moses, et al., 2005). Welch's method is one of the most widely used. The process is as follows:

1. Partition the data into multiple segments of a fixed length. The segments have a 50% overlap (this can be adjusted).

2. Each segment is multiplied by a window function (we chose to use a Blackman-

Harris window function, but found that the window function makes no noticeable difference in the results with the number of data points we had). The windowing functions are less useful for measuring the power spectrum of white noise, but must be selected carefully for measuring signal peaks (Harris, 1978).

3. The window-multiplied segment is then FFTed and properly normalized for the windowing segment used (1/(sample rate x sum(window^2)).

4. All of the FFTed segments are first squared then averaged together, and then the high frequency ranges are averaged (for us, between 100 and 900 kHz), and divided by the number of frequency points to get the PSD. The square root of the PSD at this point is the NSD.

5. The theoretical noise contributions from the circuit components can be subtracted from the PSD (always subtract power, not voltage) to get the noise attributable to the source resistance.

6. If an offset noise from unknown sources is present, this is subtracted from the PSD at this point, and reported.

Note that different programs normalize FFTs differently. For instance, FFTW normalizes its FFTs differently than Mathematica. Additionally, if a 1-sided versus 2-sided FFT is generated by the program, the result must be multiplied by a factor of 2. We strongly suggest generating a test file with generated noise to double check code, as Welch's method, implemented in slightly different ways, can be used to measure PSD with units in V^2/Hz or just V^2. The Blackman-Harris window function we used is

$$w(n) = a_0 - a_1\cos\left(\frac{2\pi}{N}n\right) + a_2\cos\left(\frac{2\pi}{N}2n\right) - a_3\cos\left(\frac{2\pi}{N}3n\right), \qquad (5.1)$$

where $n = 0, 1, 2, ..., N-1$, where N is the number of data points per segment, $a_0 = 0.35875$, $a_1 = 0.48829$, $a_2 = 0.14128$, and $a_3 = 0.01168$.

C and the FFTW libraries (Frigo and S. Johnson, 2005) are ideal for this sort of data analysis as the use of pointers makes the process less memory intensive. The FFTW allows you to make plans for an array of a certain length accessed by a pointer, which can measure the fastest method for doing a transform. This array can then be

overwritten with each successive segment data for fast processing. Generally, for one temperature data point, 500 million data points were taken and saved, and C was far superior at manipulating such large amounts of data.

Determining thermoelectric voltage

The following process was used to determine the amplitude of the thermoelectric voltage produced in our devices:

1. Read in the binary file
2. FFT the data and find the magnitude of the signal at the chopper frequency
3. Multiply the magnitude of this signal by the gain of the circuit at this frequency and the sine - to - square wave conversion factor to get the thermoelectric voltage amplitude.

The FFT was taken using either the entire length of the data set (1e8 data points per data set) or $2^{26} = 67108864$ data points if speed is an issue (FFTs are faster if the number of data points is a power of 2). This number of points gave a very high frequency resolution, and the magnitude of the FFT signal was taken to be the maximum point within about 5 or 10 Hz of the chopper signal, as the chopper was often a fraction of a Hz off the set value, attributable to the precision of the chopper.

To ensure the capacitors in the circuit had been charged and equilibrium temperature was reached, the device was chopped slowly, at around 43 Hz. As we can see from Figure 4.4, the AC coupling to the sample attenuates low frequency signals below approximately 500 Hz. Therefore, finding the thermoelectric voltage amplitude is not entirely straightforward to extract, which we will describe presently.

We start with several assumptions. First, that the incident laser light is approximately a perfect square wave, and the induced heating in the circuit is approximately a perfect square wave. From (Mauser et al., 2017), (or an earlier chapter), a similar device has a response time of about 3 kHz, which is much greater than 43 Hz, so this should be a safe assumption. Next, we know that any piecewise smooth periodic function can be written as a Fourier series. A function is piecewise smooth on some interval if the interval can be broken up into pieces such that in each piece the function is continuous and its derivative is continuous (Haberman, 2004). A finite number of jump discontinuities are allowed. From (Haberman, 2004), the Fourier series of $f(x)$ over the interval $-L \leq x \leq L$ is defined to be the infinite series

$$\text{Fourier series} = a_0 + \sum_{n=1}^{\infty} a_n \cos\frac{n\pi x}{L} + \sum_{n=1}^{\infty} b_n \sin\frac{n\pi x}{L} \sim f(x), \tag{5.2}$$

where the Fourier coefficients are given by

$$a_0 = \frac{1}{2L}\int_{-L}^{L} f(x)dx, \tag{5.3}$$

$$a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\frac{n\pi x}{L}dx, \tag{5.4}$$

$$b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\frac{n\pi x}{L}dx. \tag{5.5}$$

We use $\sim f(x)$ as the infinite Fourier series doesn't always converge and if it does, it doesn't always converge to $f(x)$. For most physical problems, the Fourier series should converge to $f(x)$. Conveniently, this is the case for a square wave with a 50% duty cycle, $g(t)$, which has the following Fourier series

$$g(t) = \frac{4A}{\pi}\sum_{n=1,3,5,\ldots}^{\infty}\frac{1}{n}\sin n\omega t, \tag{5.6}$$

where the wavelength is $2L$ and A is the amplitude ($A = \frac{1}{2}V_{pp}$).

Further, from (Haberman, 2004), If $f(x)$ is piecewise smooth, then the Fourier series of a continuous function $f(x)$ can be differentiated term by term if $f(-L) = f(L)$, which is true in the case of a periodic square wave. This is relevant, as our AC-coupled circuit is called an "RC differentiator circuit," where the voltage seen by the opamp across R_3 in Figure 4.2 can be found by first using Kirchoff's voltage law

$$V_{in} = \frac{Q}{C_1} + IR_3, \tag{5.7}$$

where V_{in} is the voltage (relative to ground) just to the left of C_1 in Figure 4.2, and Q is the charge on C_1. We can differentiate this to

$$R_3 \frac{dI}{dt} + \frac{I}{C_1} = \frac{dV_{in}}{dt}. \tag{5.8}$$

We then set $V_{in}(t) = g(t)$ in Equation 5.6 above, but for convenience, change to complex notation so $V_{in}(t) = \frac{2A}{i\pi} \sum_{n=\ldots-3,-1,1,3,\ldots} \frac{1}{n} e^{in\omega t}$ and $I(t) = \sum_{n=\ldots-3,-1,1,3,\ldots} \frac{I_n}{n} e^{in\omega t}$. We assume I_n is complex. Then,

$$R_3 \sum_{n=\ldots-3,-1,1,3,\ldots} I_n i\omega e^{in\omega t} + \frac{1}{C_1} \sum_{n=\ldots-3,-1,1,3,\ldots} \frac{I_n}{n} e^{in\omega t} = \frac{2A}{\pi} \sum_{n=\ldots-3,-1,1,3,\ldots} \omega e^{in\omega t}. \tag{5.9}$$

Due to the orthogonality of the exponents, we can drop the sums, and cancel the exponents, leaving

$$I_n \left(R_3 i\omega + \frac{1}{C_1 n} \right) = \frac{2A}{\pi} \omega. \tag{5.10}$$

Rearranging,

$$I_n = \frac{\frac{2A}{\pi}\omega}{\left(R_3 i\omega + \frac{1}{C_1 n} \right)}. \tag{5.11}$$

Thus, $V_{out_n} = \mathrm{Im}[I_n R_3]$ (since we only want the sine components), where V_{out_n} is the amplitude of the nth harmonic at the output, and is

$$V_{out_n} = \left| \frac{2n\omega C_1}{\pi(1 - (R_3 \omega C_1 n)^2)} \right| A, \tag{5.12}$$

which shows that the amplitude of the input voltage can be theoretically determined from the amplitude of the output voltage for each harmonic of the Fourier series, if we solve for A. More succinctly, differentiation is a linear operation. Note that

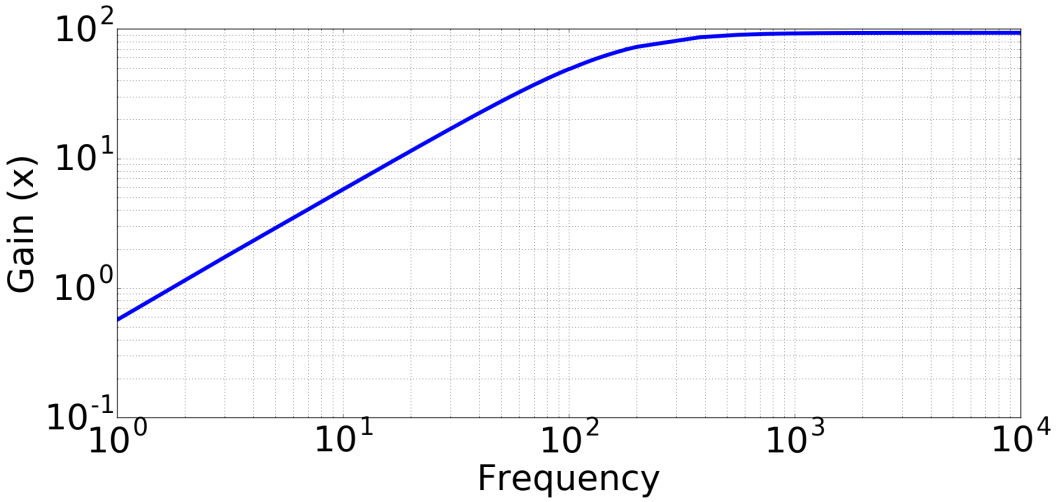

Figure 5.2: Measured gain versus frequency on our 2-stage opamp circuit. Note the similarity to theory in Figure 4.4.

Equation 5.9 actually has two infinite sums, due to differentiating a non-continuous Fourier series. Clearly this is not physical, and a real function is continuous and not a perfect square wave, so there will be a truncation of terms.

For greater accuracy, we measured V_{out_n} as a function of frequency for fixed A to determine this attenuation factor. We connected the external reference generator of a lock-in (a sine wave) to a point in the circuit between $R_s = 3300\ \Omega$ and C_1, and measured the output after both opamps with the lockin input. We swept the frequency (see Appendix for code to connect to lockin), with results shown in Figure 5.2.

In our analysis code, we multiply the magnitude of the measured FFT signal at the chopper frequency by this gain, and then by $\frac{2\pi}{4}$, to convert the sine wave amplitude into a square wave amplitude (the $\frac{\pi}{4}$ term). The factor of 2 is because we are interested in the peak-to-peak amplitude, or $2A$.

Determining resistance

The resistance changes were determined by measuring the magnitude of the voltage peak from injection of a current between 18 and 21 kHz, as described in an earlier section. To measure the magnitude of this peak, steps a) and b) depicted in Figure 5.1 were enacted on the data, and then each individual section of "on" and "off" data was Fourier transformed and the magnitude at the frequency of interest was probed. We did not proceed to step c) in Figure 5.1, as "squishing" together all of the on and

off data would create a wave discontinuous phase (Figure 5.3), the Fourier transform of which would not give the desired amplitude. Fourier transforming each "on" and "off" section of data individually gives less frequency resolution due to fewer data points and therefore may underestimate the resistance peak, but this can be mitigated by using no windowing function when Fourier transforming the data, which will give poor frequency resolution, but good magnitude estimation. As we care more about magnitude than frequency resolution, this is acceptable.

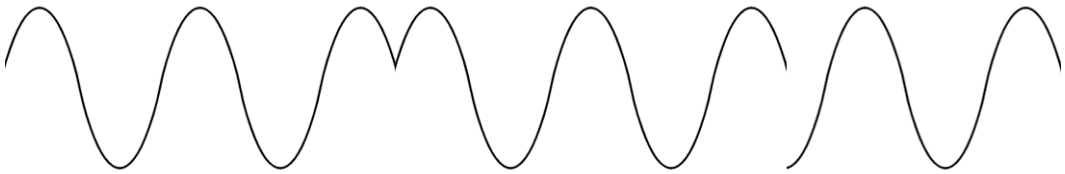

Figure 5.3: A sine wave with piece-wise discontinuous phase.

5.2 Noise thermometry circuit measurement verification

To test the noise thermometry circuit, we attached metal film resistors in place of the sample. Metal film resistors have noise properties close to theoretical values. The circuit was placed in a sealed styrofoam box (Figure 5.4) with a resistive heater (Figure 5.5). A constant DC current was run through the resistive heater, and the system was allowed to thermalize for about an hour, after which time 500 million data points at 2 MS/s were collected from each of the 4 opamps systems in the circuit. Then the DC current through the resistive heater was raised, and the system thermalized for an hour, and the process was repeated. Figure 5.6 shows the inside of the metal box in Figure 5.5.

To test the noise thermometry circuit, we first put 4 different 1% metal film resistors in each of the four channels. In the first round of testing, we used 3.3 kΩ, 4.7 kΩ, 6.8 kΩ, and 10 kΩ each in a different channel. In the second, third, and fourth round we rotated these resistors among each opamp. Figure 5.7 show 6.8 kΩ (top) and 10 kkΩ (bottom). Figure 5.8 shows 3.3 kΩ (top) and 4.7 kΩ (bottom).

In the next round, we used smaller resistances: 1.5 kΩ, 2.2 kΩ, 3.3 kΩ, and 4.7 kΩ. We redid the latter two resistances to examine the stability of our circuit after about a week. Figure 5.9 shows 3.3 kΩ (top) and 4.7 kΩ (bottom), measured about a week after the data taken in Figure 5.8. Figure 5.10 shows 1.5 kΩ (top) and 2.2 kΩ (bottom).

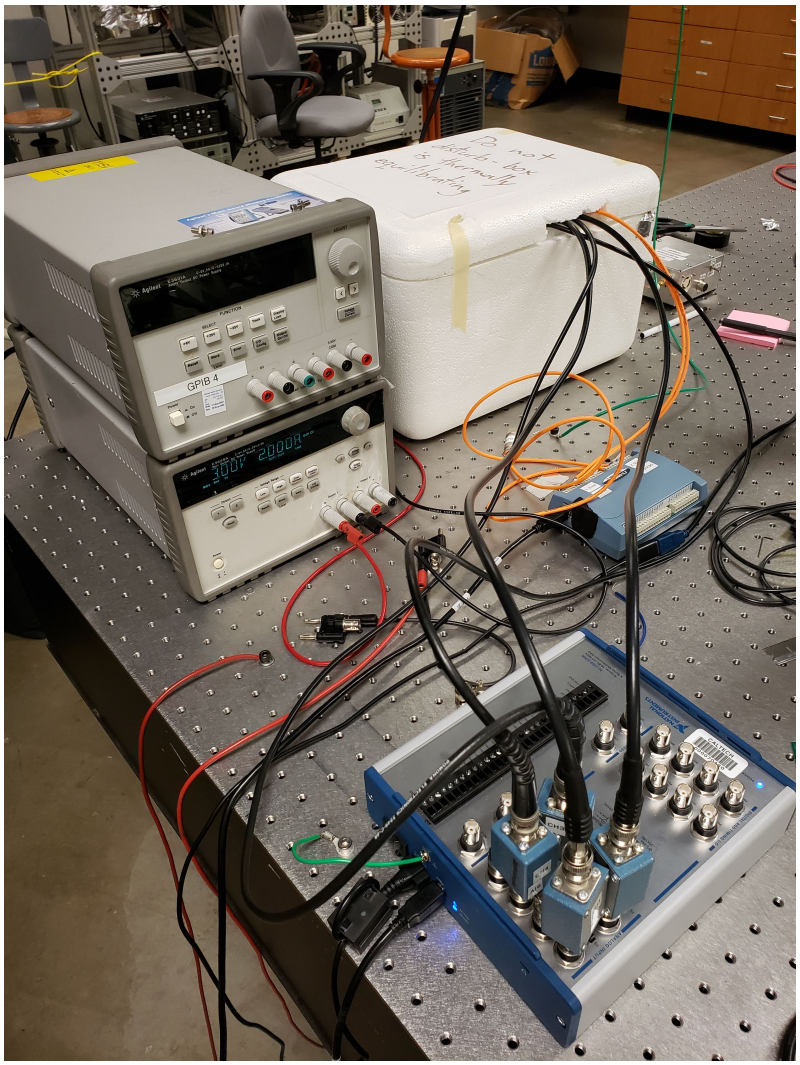

Figure 5.4: The styrofoam box is holding the noise thermometry circuit. The small blue box with orange twisted shielded wires is connected to the surface mount RTDs on the circuit is a DAQ from Omega described above for logging temperature data. The power supply on the left is connected to the resistive heater in the box. The instrumetn in the foreground is the NI-USB 6366 DAQ AC-coupled to BNC cables connected to the output of the 2nd stage opamp circuit.

We notice several trends from the data. First, there is an offset from theory that is apparent in the plots. This is attributable to unaccounted for noise sources, such as errors in opamp data sheets. The slope of the theory and experiment is similar, especially at higher resistances, leading us to believe our noise thermometry circuit is capable of measuring temperature. Deviations from the theoretical slope could be due to error in the 1% resistor values, as well as temperature measurement errors from the on-board RTD. The RTD has accuracy of 0.15 K. The circuit was designed

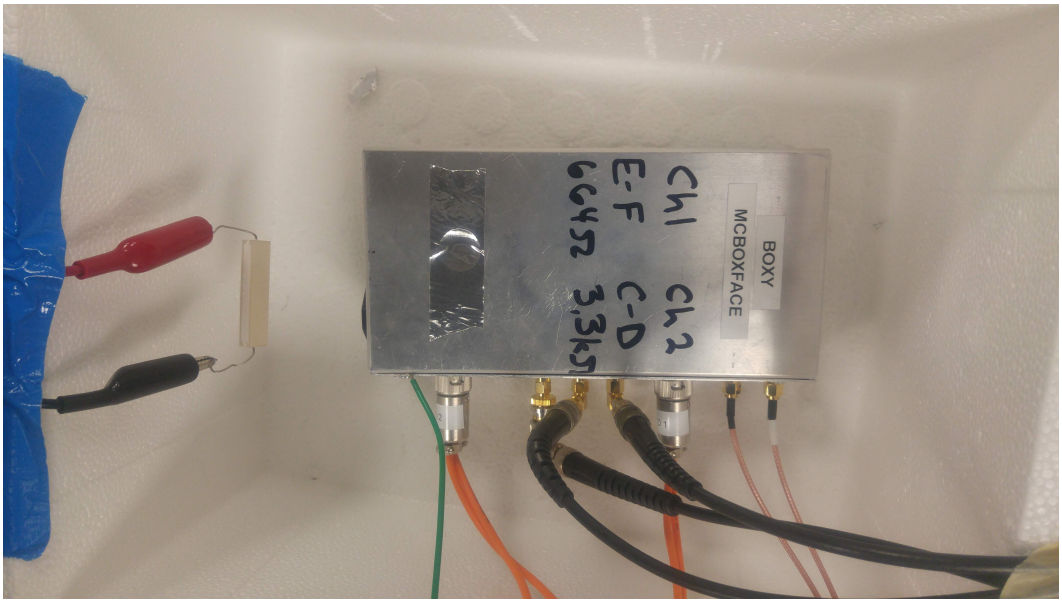

Figure 5.5: The inside of the cooler. The left side shows the suspended resistive cooler. The circuit is in the shielded metal box labeled "Boxy McBoxFace". The green wire is a grounding wire for the shielding.

with channel 4 signal traces too close to the other on-board RTD (the one close to the sample), so we were forced to leave it unhooked to not interfere with the measurements. This RTD would have been a slightly more accurate measurement of the resistor temperature, as it was several centimeters closer, in case the thermal stability time of 1 hour was not long enough. Figure 5.11 shows an expanded view of the measurements from one circuit.

System temperature and precision

The system temperature was determined using the following method. To approximate the system temperature, we used data from the previous section. We took the PSD data from the output of the DAQ and divided out the gain. We then plotted a best fit line to this data, and followed the data back to the point at which it crossed the x-axis. The absolute value of this x-value is the system temperature. Figures 5.12, 5.13 shows this data for different resistor values.

The precision of the system was measured and compared to Dicke's radiometry equation, shown in Figure 5.14. The orange line is the prediction from Dicke's equation.

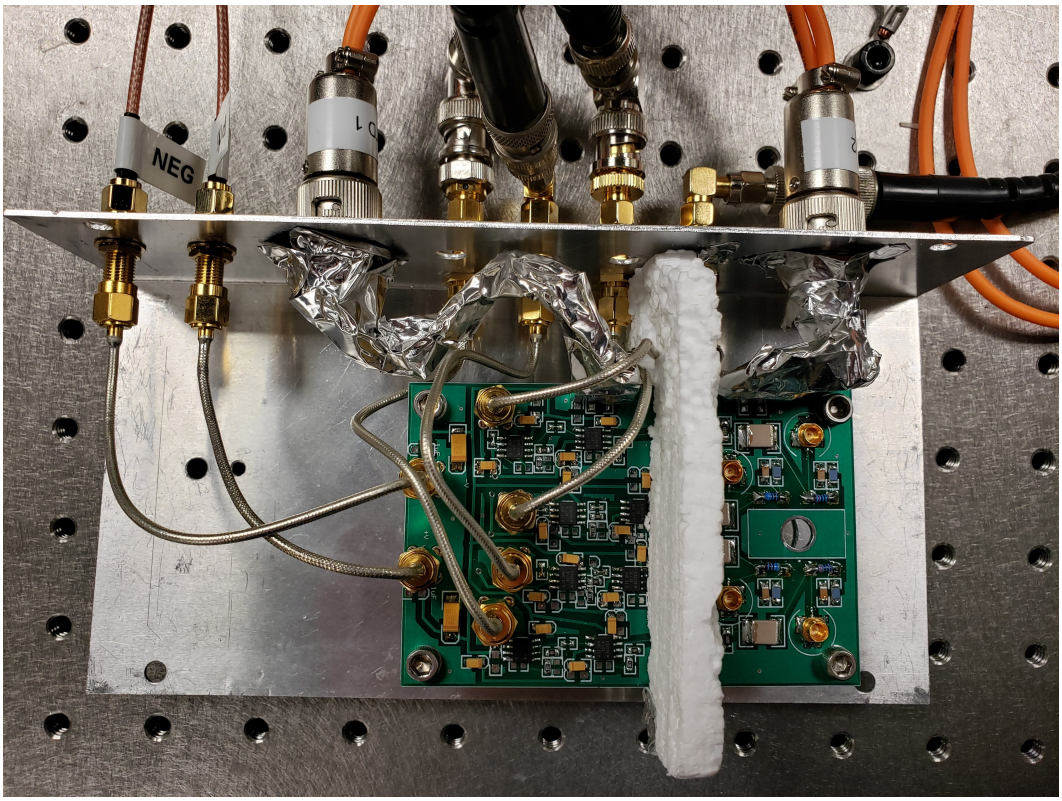

Figure 5.6: The inside of the shielding box. The syrofoam divider is to further thermally isolate the sample from opamp heating. The samples, on the right, are metal film resistors. The cables connected to the circuit, from left to right, are as follows: shielded SMA cable for -5V power supply from the voltage regular box; the matching +5 V power supply; the orange cable is shielded twisted wires taking 4-wire RTD measurements from the SMT RTD; the next 4 cables are shielded BNC cables carrying the opamp output to the USB 6366 DAQ; and the far right cable is another SMT RTD cable.

5.3 Noise measurements under chopped illumination

The setup consists of either a broad-spectrum laser passed through a monocromator to filter out the wavelength of interest or a diode laser, depending on the power requirements. The light passes through a chopper and then a glass slide (~10/90 beamsplitter), which is used to monitor the stability of the laser power with time. The light passes through various alignment mirrors then through the sample to a photodiode behind the sample, which monitors any shifts in the sample position with time.

We first plot the PSD as a function of absorbed laser power for a 639 nm diode laser, with electric field polarized perpendicular to the wires, and a chopper frequency of 43 Hz. At this wavelength, the absorption in the wires was approximately 17%.

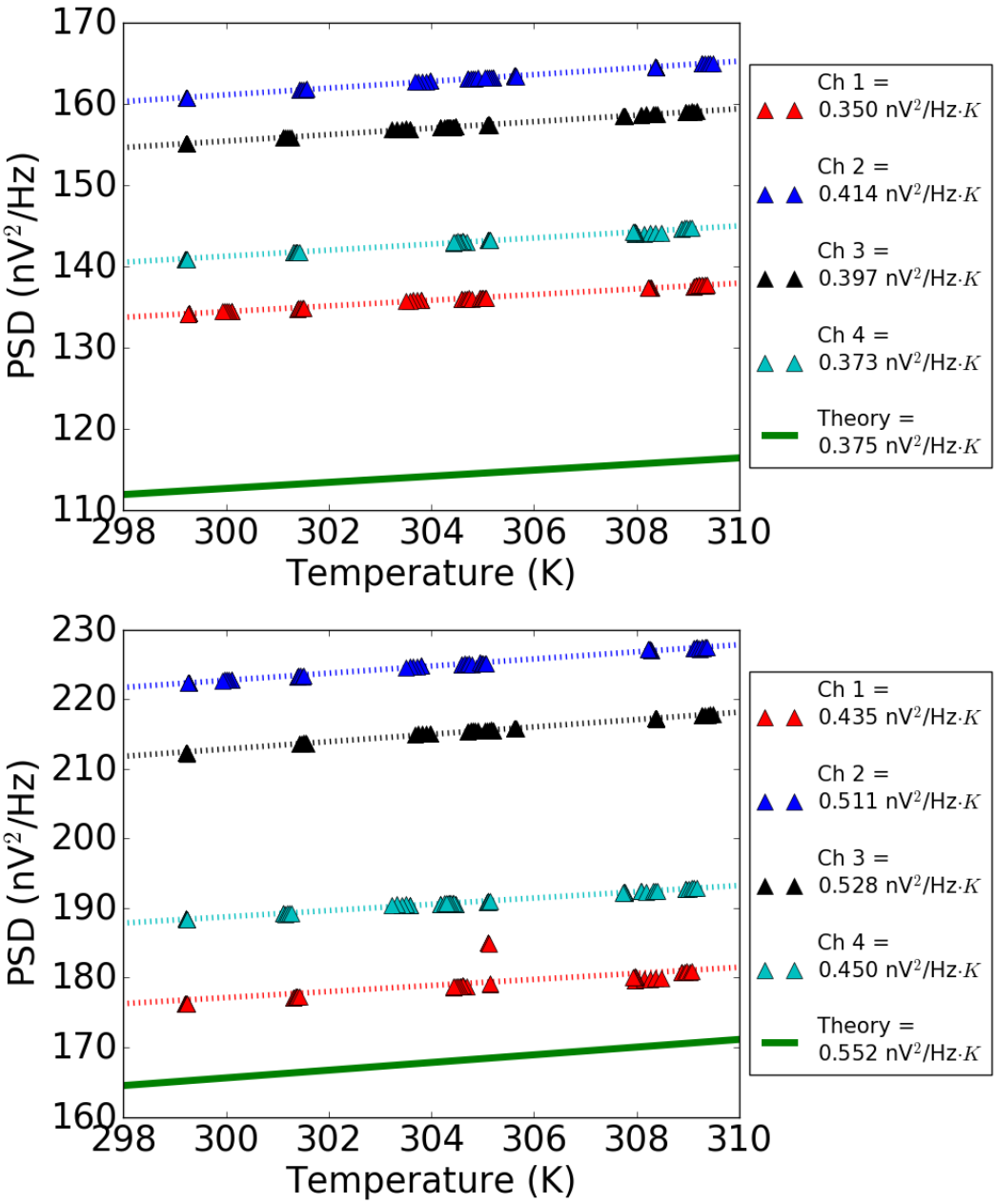

Figure 5.7: Test of the circuit with metal film resistors, with values of 6.8 kΩ (top) and 10 kΩ (bottom). The channel refers to which op amp on the circuit did the measurement. The legend shows the slope of a best fit line to the data.

Figure 5.15 shows PSD versus absorbed laser power assuming constant sample resistance. Blue data points are when the light is blocked by the chopper, and red data points are when the light is not blocked by the chopper. We see that the noise appears to decrease when the light is off, which is unexpected, and can be explained

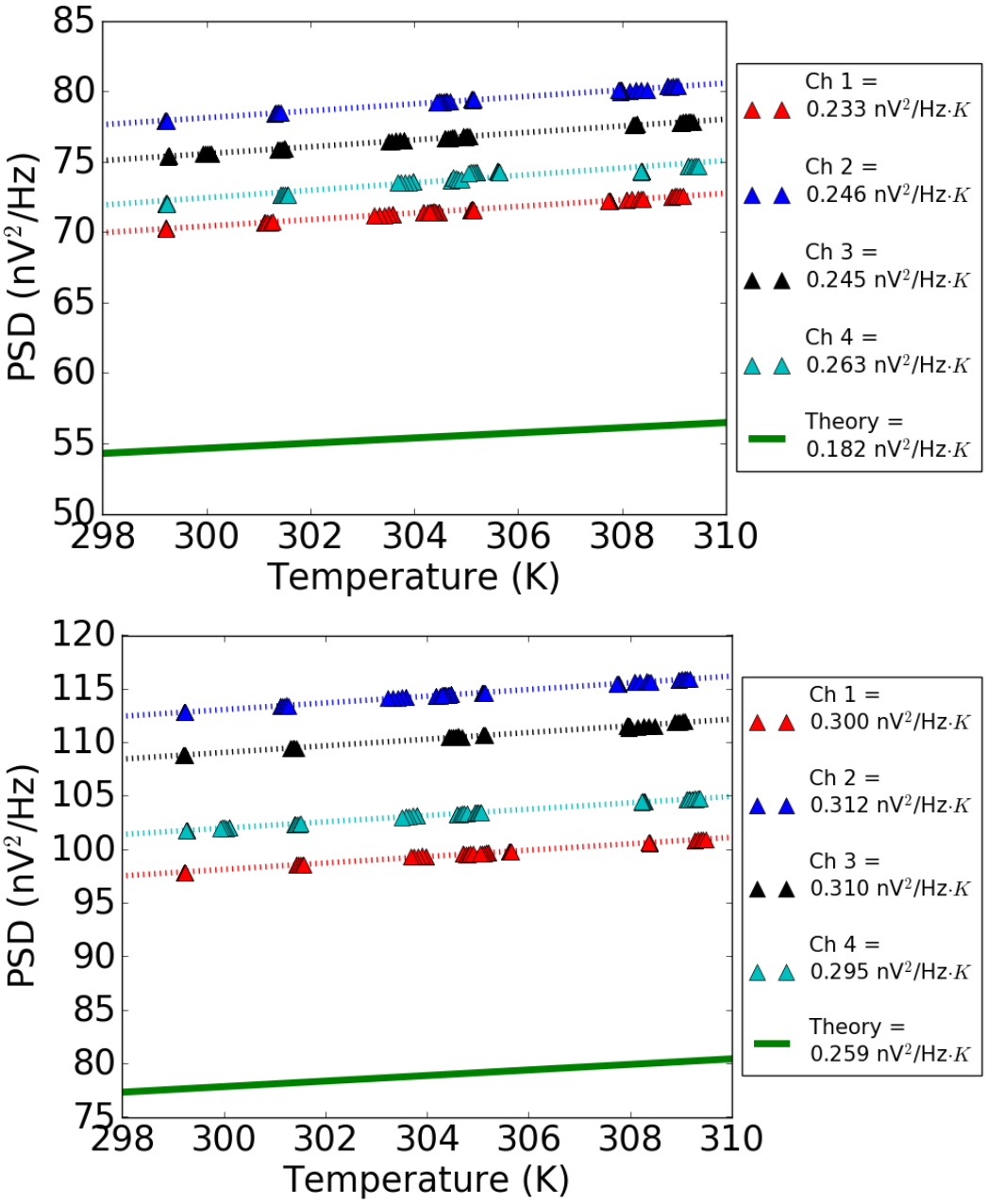

Figure 5.8: Test of the circuit with metal film resistors, with values of 3.3 kΩ (top) and 4.7 kΩ (bottom). The channel refers to which op amp on the circuit did the measurement. The legend shows the slope of a best fit line to the data.

either by decreasing temperature or decreasing resistance of the sample.

The resistance of the sample as a function of absorbed power is plotted in Figure 5.16. The dotted line and greed arrow indicate the order in which measurements were

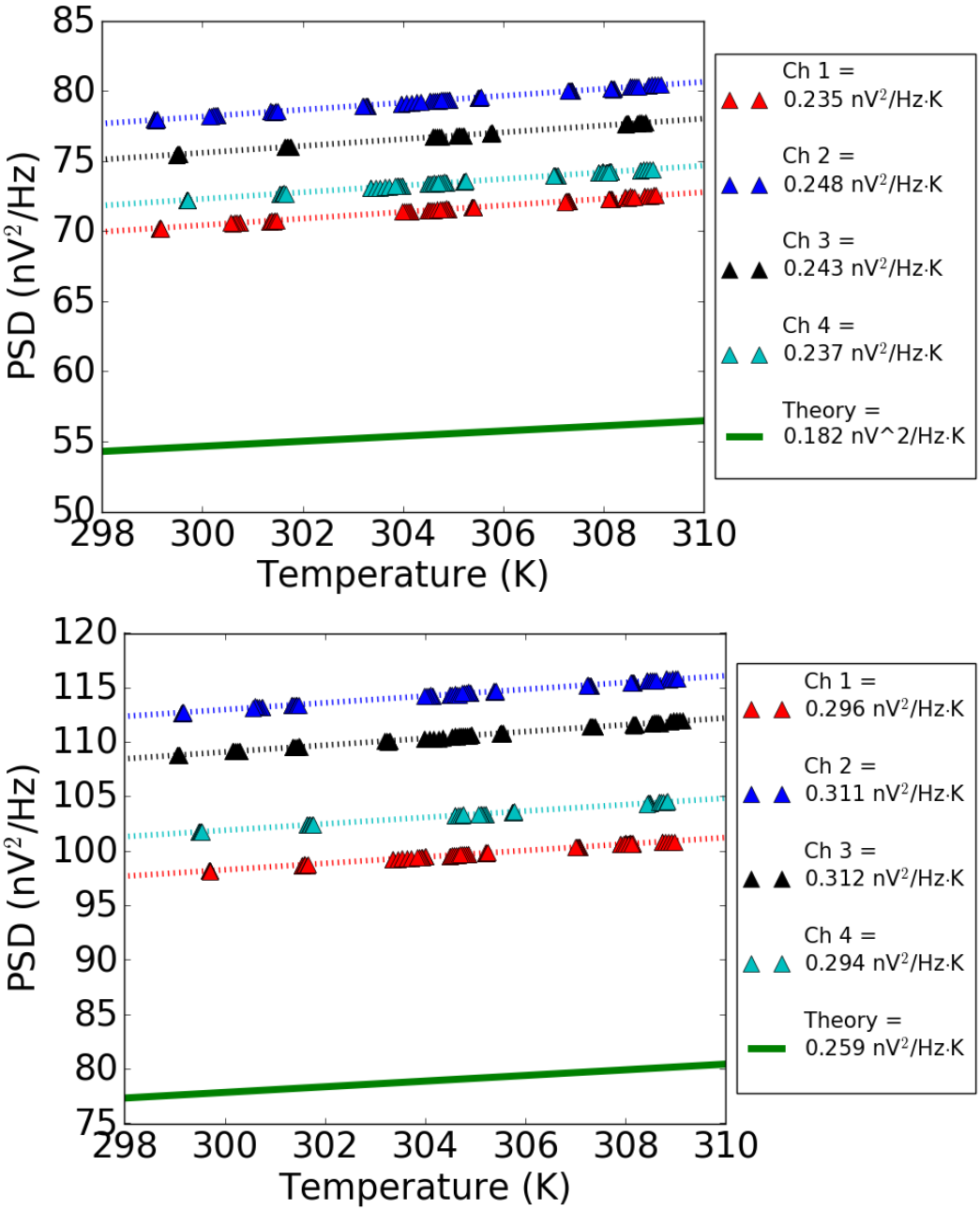

Figure 5.9: Test of the circuit with metal film resistors, with values of 3.3 kΩ (top) and 4.7 kΩ (bottom). These measurements were taken about a week after those shown in Figure 5.8. The channel refers to which op amp on the circuit did the measurement. The legend shows the slope of a best fit line to the data.

taken. This plot offers two important insights: first, that the resistance decreases with increasing power absorption, and second, the resistance does not return to its original value after illumination. Addressing the first point, we had expected that because the

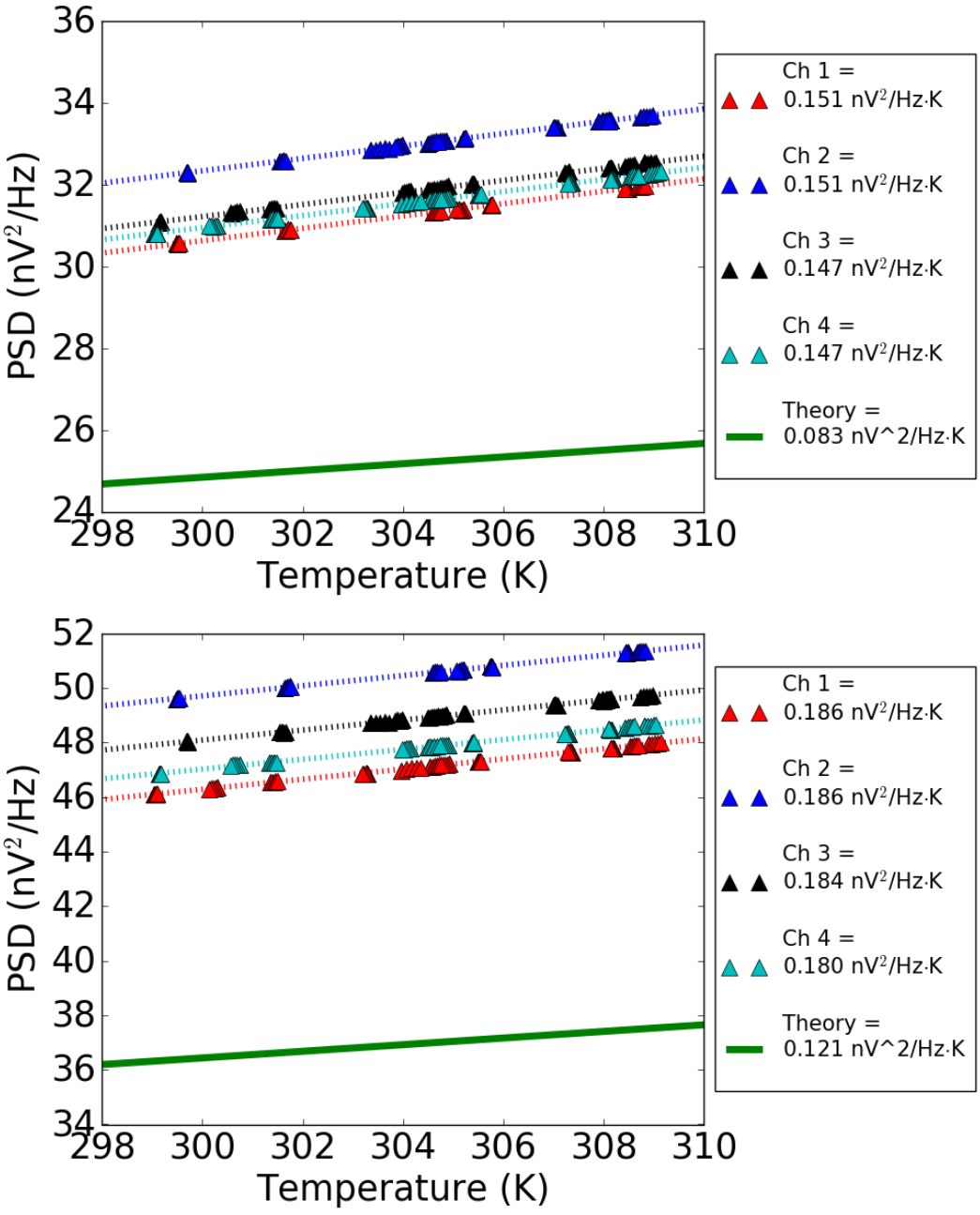

Figure 5.10: Test of the circuit with metal film resistors, with values of 1.5 kΩ (top) and 2.2 kΩ (bottom). The channel refers to which op amp on the circuit did the measurement. The legend shows the slope of a best fit line to the data.

carrier concentration of bismuth telluride is so high ($10^{19} - 10^{21}/\text{cm}^3$), the resistance would increase with temperature (assuming that illuminating the sample heats it), similar to a metal. Instead, bismuth telluride acts more like a semi-conductor, in that

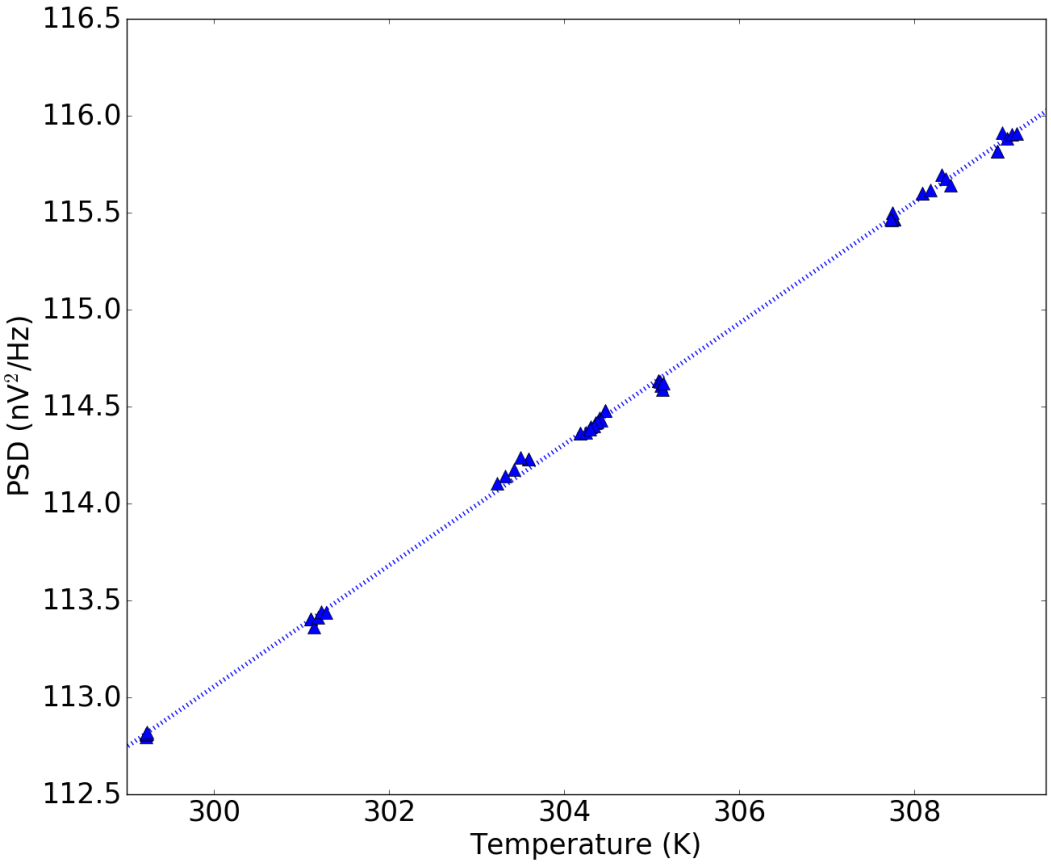

Figure 5.11: Plot of only channel 2 in Figure 5.8 (bottom), with resistance value of 4.7 kΩ. The dotted line is the best fit line.

temperature (or potentially photo-excited electrons) decrease resistance. As for the second point, the resistance of bismuth telluride does not appear stable with time. This could be due to the cyclic annealing of the material under chopped illumination altering the crystal structure. This indicates that bismuth telluride would not be a good candidate for resistive thermometry, where the temperature of a material is determined by change in resistance.

Putting the noise and resistance data together, we find the temperature as a function of absorbed power, shown in Figure 5.17, across two different leads. We assumed a constant offset of noise (subtracted in quadrature), found by matching the on-board RTD temperature with the no-illumination data points, incorporating the resistance corrections. For the leads closer to the wires or point of illumination, the subtracted offset was 3.44 nV/$\sqrt{\text{Hz}}$, and for the leads farther from the wires or point of illumination, the offset was 3.14 nV/$\sqrt{\text{Hz}}$. Incorporating resistance into

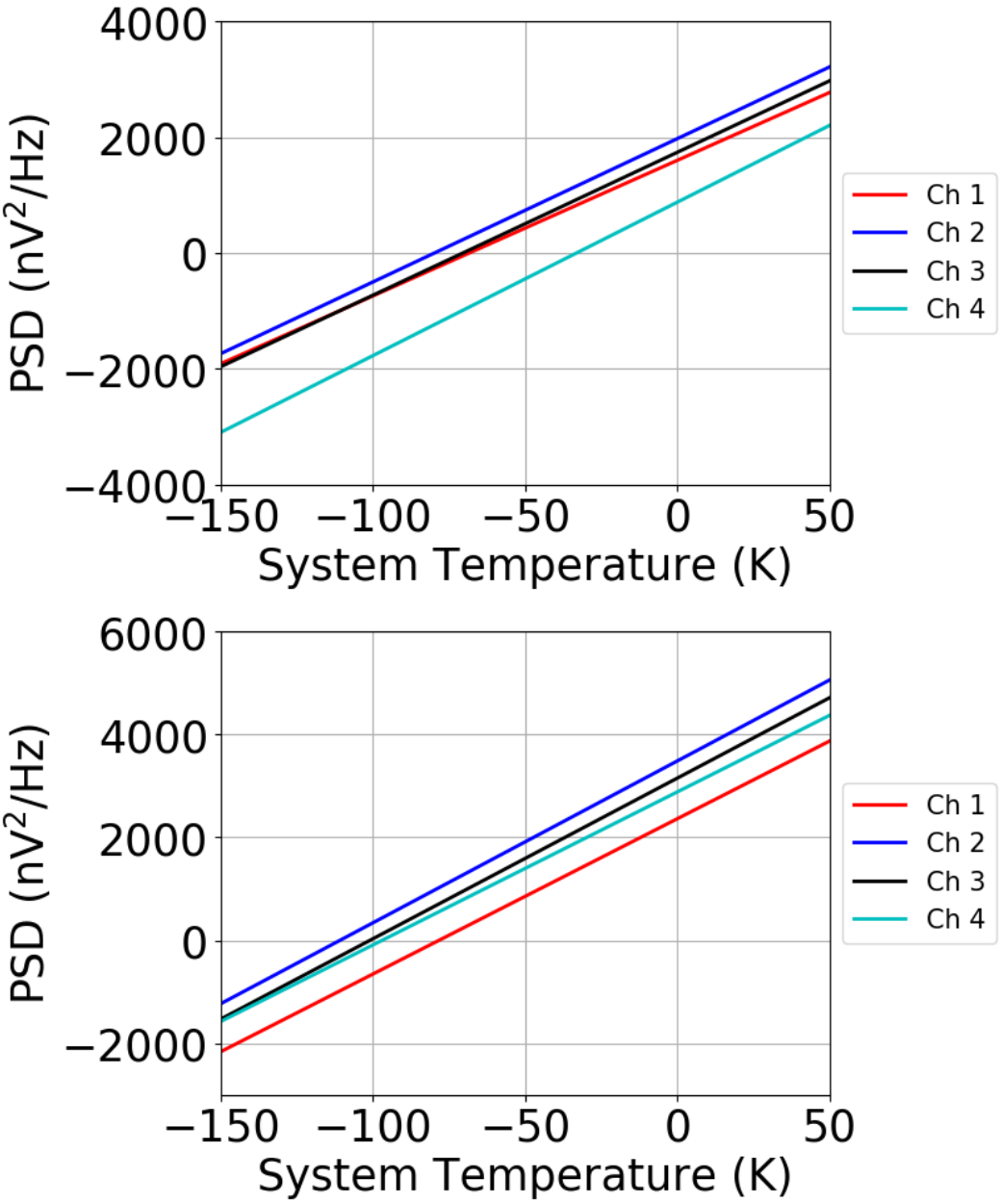

Figure 5.12: Plot of system temperature, with resistors of values 3.3 kΩ (top) and 4.7 kΩ (bottom). The channel refers to which op amp on the circuit did the measurement.

the noise measurements, we find that the temperature with the light off drops to room temperature for every power, indicating that the thermal time constant of the device is much faster than our chopping speed, which is expected based on results from our hyperspectral light detecting structure from a previous chapter. Further,

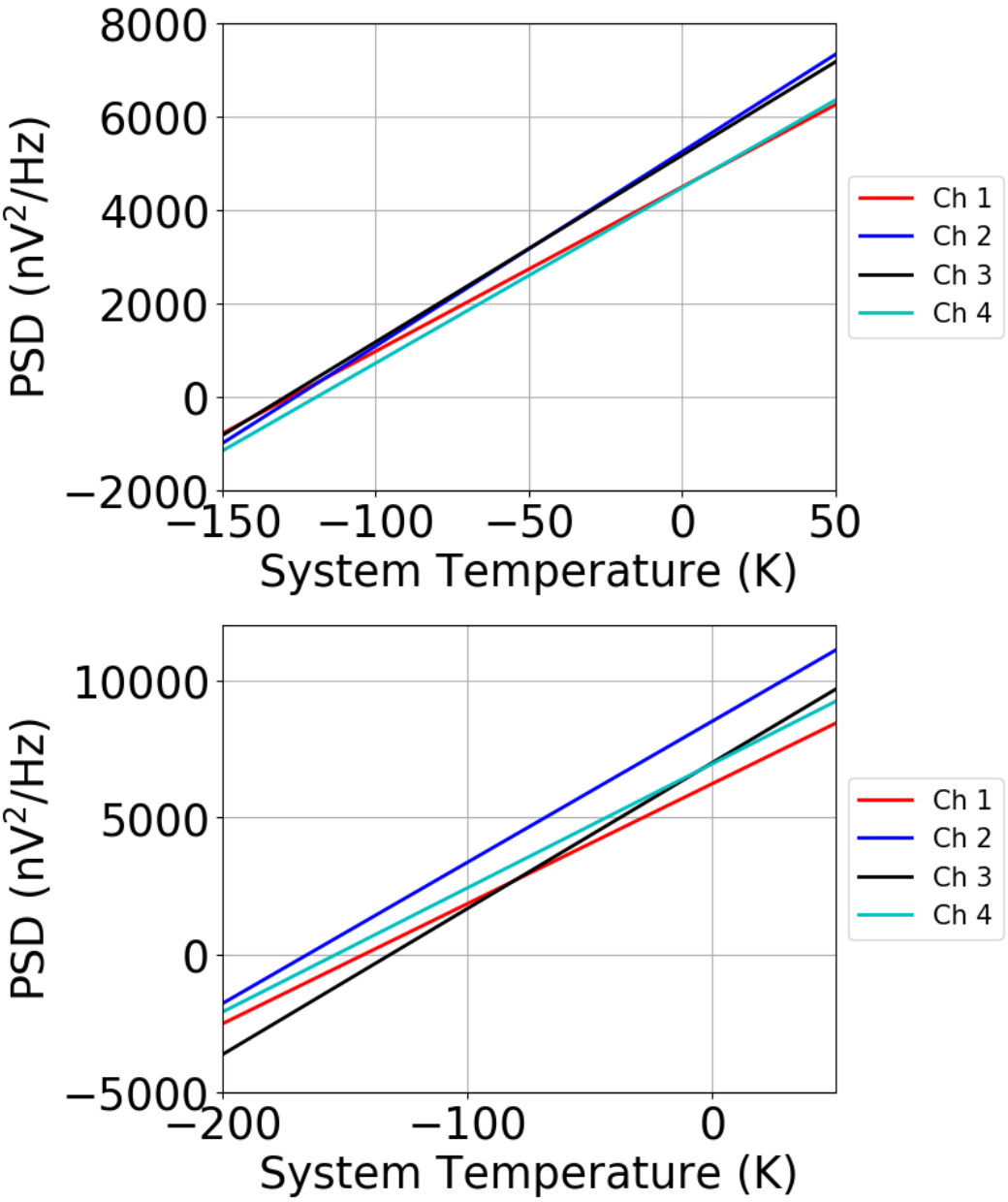

Figure 5.13: Plot of system temperature, with resistors of values 6.8 kΩ (top) and 10 kΩ (bottom). The channel refers to which op amp on the circuit did the measurement.

the temperature increases approximately linearly with increasing absorbed power, and reaches as much as 70 K above the no illumination condition, for the maximum power we used.

We can calculate the Seebeck coefficient for our bismuth telluride by dividing the

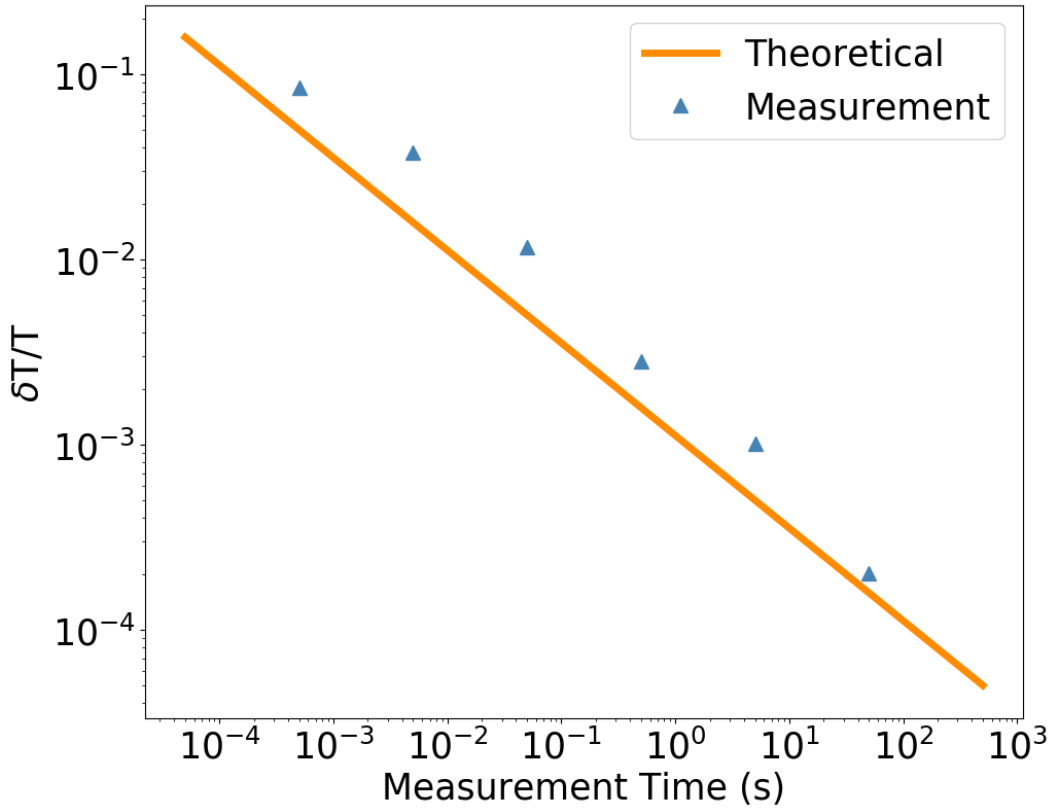

Figure 5.14: Precision of temperature measurements as a function of sampling time, for a 2 MHz sampling rate. The theoretical line (orange) is given by Dicke's equation, and the measurements (blue) are for a 1500 Ω source resistor. δT is the standard deviation of 10 measurements of NSD for the given measurement time, without incorporating a temperature offset correction. The on-board RTD reported a temperature of 298.2 K for the measurements.

measured thermoelectric voltage (the voltage difference between each lead pair when the light is on) by the temperature difference between each lead pair when the light is on. The results are shown in Figure 5.18 (left). The bismuth telluride has a Seebeck coefficient around -140 μV/K, making it n-type. We note that the Seebeck coefficient is relatively constant with temperature, as is expected over our relatively small temperature range. We find a Seebeck coefficient of zero at zero power, as the voltage difference is zero.

5.4 Outlook and future work

We were able to design and build a circuit and data analysis program that worked effectively as a primary temperature measurement system and was also able to measure thermoelectric voltages, at relatively low cost. Experimentally, the circuit

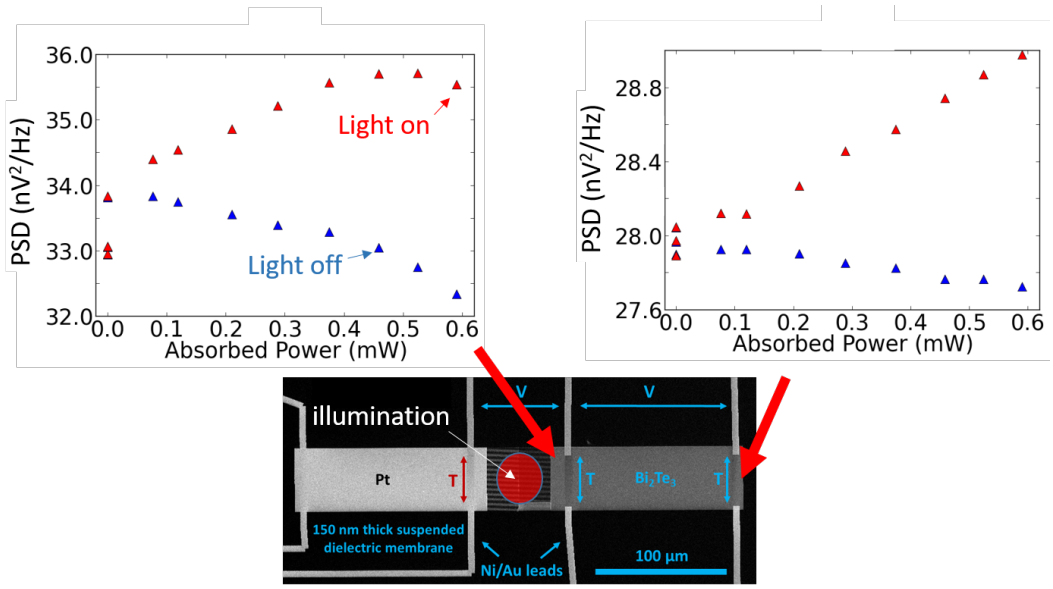

Figure 5.15: Shown is PSD under laser illumination without accounting for sample resistance.

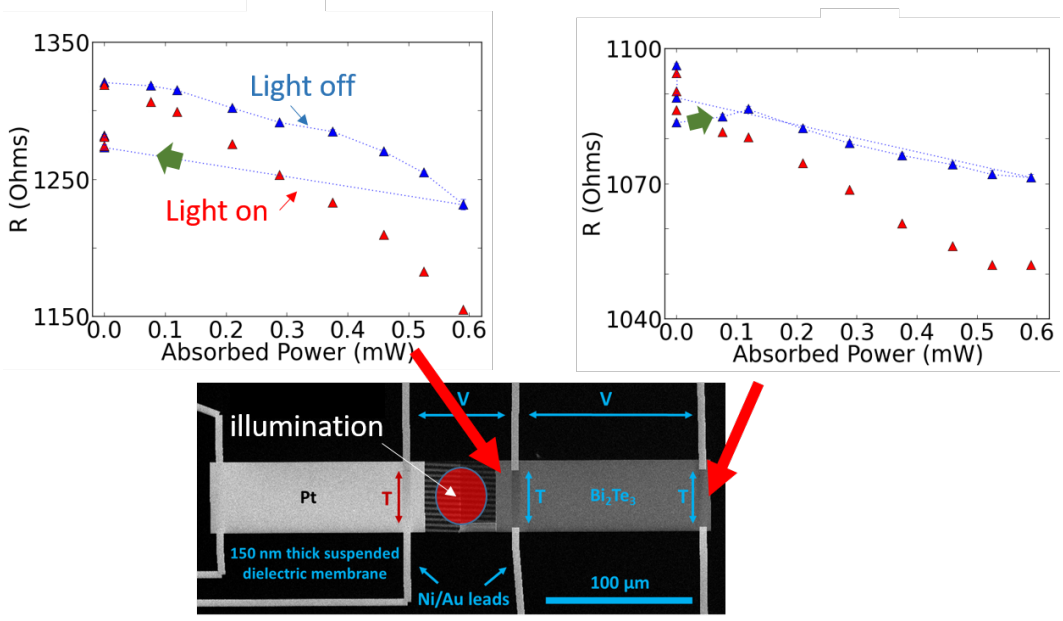

Figure 5.16: Shown is resistance of sample as a function of absorbed power.

was an effective noise thermometer for metal film resistors. Results thus far show successful measurements of temperature, voltage, and thereby, Seebeck coefficients of nanophotonic-heated structures. Further work will use these findings in addition to thermoelectric voltage measurements between the platinum and bismuth telluride leads to measure the temperature rise remotely at the center of the wires, far from

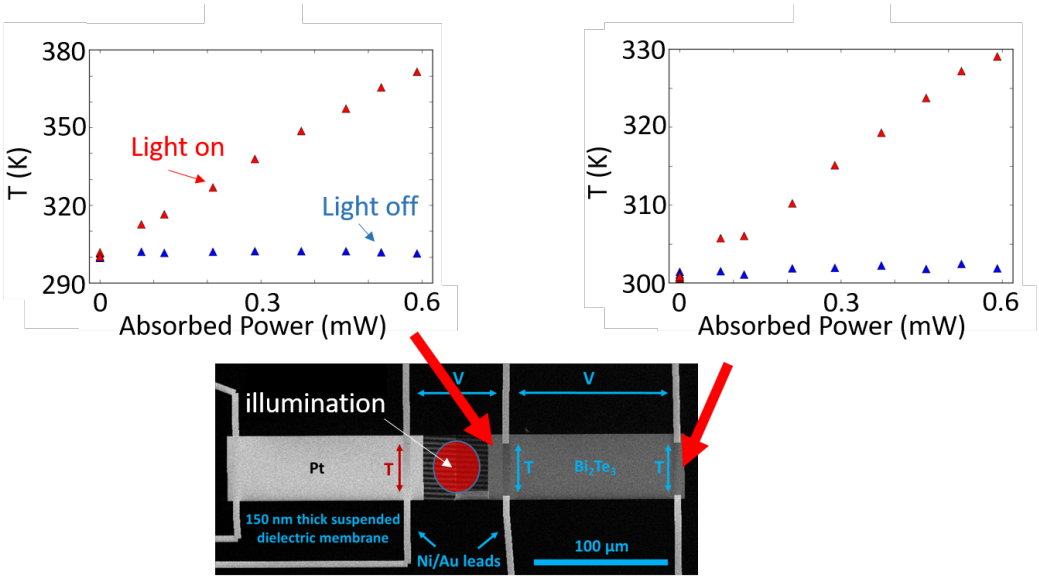

Figure 5.17: Shown is the measured Temperature of the sample as a function of absorbed power, correcting for sample resistance.

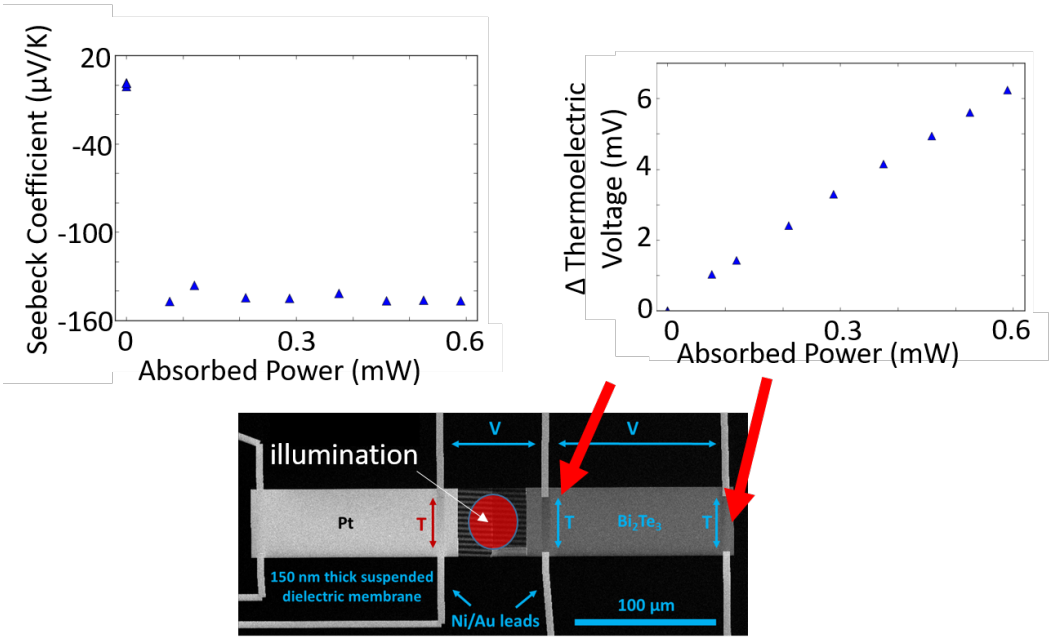

Figure 5.18: Shown is the measured Seebeck coefficient (left) for our bismuth telluride, using the data from Figure 5.17 and the measured thermoelectric voltage (right).

the leads.

Noise measurements combined with the thermoelectric effect show potential for non-

invasive measurements of temperature of nanostructures. While leads are required to connect to the nanophotonic structure, this could be a better option that needing to couple in a laser (Raman thermometry) or use a microscope (SThM, FTIR). Additionally, because we have a low noise circuit, high-zT thermoelectric materials don't necessarily need to be used in measurements. The low voltages produced by small Seebeck coefficients in gold, for instances, could be measured with our circuit, as long as they are above the noise floor of about 4-5 nV. Alternative circuits with different opamps could be designed to work more effectively for low resistance sources, as described in earlier chapters.

Chapter 6

CONCLUSION

6.1 Potential for nanophotonic thermoelectric detectors

This work combines the often disparate fields of nanophotonics and thermoelectrics by using high light-confinement to control temperature gradients in nanoscale volumes to produce highly wavelength-dependent thermoelectric voltages from spatially-uniform illumination. This wavelength dependence is geometrically tunable and enables a non-bandgap limited, uncooled, filterless two-material spectrometer from visible to infrared on one chip. Because of the small volume, a rise time 100 times faster than conventional thermoelectric detector was found. Thermopiling, optimized thermoelectric materials, or thermal management via wire suspension would improve responsivity, while shrinking absorber dimensions would decrease response time. Extending this thermoelectric detection motif to other resonant nanophotonics structures will be straightforward, as we briefly explored in our work, and opens up a new, untested world of uncooled self-filtering light detecting structures.

Designing thermoelectric nanophotonic detectors involves a vast parameter space, involving not only nanophotonic design, but thermal design and thermoelectric properties of materials used. For higher responsivity, larger temperature gradients are required, which favors long and thin structures. For faster response time, smaller structures with lower heat capacities win out, which makes device design tricky, as these are contradictory requirements. One interesting area for these thermoelectric nanophotonic detectors would be in the ultra-violet regime. Here, nanophotonic structures naturally shrink to handle smaller wavelengths, which would decrease the response time. Environments with higher background radiation, such as nuclear or space applications, could also find uses for these detectors, as thermoelectrics are less sensitive to radiation than photovoltaic detectors. An interesting study would be on the optimization of thermoelectric nanophotonics for speed and responsivity, and to understand if these have absolute limits.

More thermal modeling is needed to determine the absolute detectivity limit of these thermoelectric detectors. This is limited in part by the approximately 7% efficiency ceiling of currently usable thermoelectric materials. Nanophotonic devices have

been designed which can absorb nearly 100% of incident light, though there will be efficiency losses through heat loss via radiation, conduction, and convection, which need to be studied in more detail (a similar study for solar thermoelectrics was performed in (Kraemer et al., 2011)). Studying cooled detectors, weighing the benefits of lower noise with a different material selection also needs to be studied.

Another interesting detector idea would be to combine resistive detectors, or bolometers, with thermoelectrics. In principle, an increase in temperature caused by an impinging photon changes the resistance of the bolometer. The addition of a thermoelectric material could give additional information about the location of the photon hitting the detector (based on the sign of the voltage), or generally enhance the signal by generating added thermoelectric voltage, which would act as additional or decreased resistance depending on the location of the photon on the structure.

6.2 Potential for noise thermometry in nanophotonic devices

By measuring the energy of the electrons themselves within a material, we are able to make primary thermometry measurements less invasively and with simpler connections to the sample than with other temperature measurement techniques. While noise thermometry may not be the best thermometry technique for all nanophotonic applications, it does show promise for very thin structures where resistive thermometry probes would be on the order of the size of the structure and would thus effect the temperature profile. Additionally, the cooling results when coupling a thermoelectric nanophotonic device to a capacitive circuit show promise for on-chip cooling applications.

One interesting path to explore would be resistance-corrected noise thermometry, or conversely, noise-corrected resistive thermometry. Resistive thermometry works well for materials with well-defined resistance versus temperature curves, but not all materials have this characteristic, as we observed in bismuth telluride, for instance, which had a resistance that drifted with time. By either monitoring resistance constantly during noise measurements, or by periodically using noise thermometry to recalibrate a resistor used in noise thermometry, any material could be used for thermometry, regardless of the stability of its resistance with time.

Noise measurements ave not been extensively used in nanophotonics, and there are many avenues to explore. For instance, to avoid metallic leads from affecting nanophotonic temperatures, capacitive coupling to nanophotonic structures could be accomplished by separating the structures from metallic leads with a thin dielec-

tric layer, effectively forming a capacitor. At high enough frequencies, capacitors have negligible, allowing for noise temperature extraction without additional noise. Measuring the noise in interferometric measurements can reflect the actual motion of electrons in a material, thereby allowing for noise temperature extraction. Similar methods could be used, for example, by diffracting light with subwavelength structures and using the noise of the diffracted pattern to determine the temperature of the nanophotonic structure.

Additionally, while noise in equilibrium systems is well studied and what we have studied here, the "transition" noise, or the noise in the heating and cooling transition regions we ignored, could have interesting information about material properties. In addition, exploring the possibility of using coupling between large capacitors, nanophotonic heating, and thermoelectric materials to generate periodic thermoelectric cooling would be an interesting direction to pursue theoretically and potentially experimentally. Finally, using measurements of power spectral density and thereby correlation functions to extract the Seebeck coefficient through Green-Kubo theory of hard-to-measure materials, such as 2D materials, would be an interesting avenue to explore.

BIBLIOGRAPHY

Adam, Elhadi, Onisimo Mutanga, and Denis Rugege (2010). "Multispectral and hyperspectral remote sensing for identification and mapping of wetland vegetation: a review". In: *Wetlands Ecology and Management* 18.3, pp. 281–296.

Alaee, Rasoul et al. (2012). "A perfect absorber made of a graphene micro-ribbon metamaterial". In: *Optics express* 20.27, pp. 28017–28024.

Amatya, R and RJ Ram (2010). "Solar thermoelectric generator for micropower applications". In: *Journal of electronic materials* 39.9, pp. 1735–1740.

Antonelli, G Andrew et al. (2006). "Characterization of mechanical and thermal properties using ultrafast optical metrology". In: *MRS bulletin* 31.8, pp. 607–613.

Atwater, Harry A and Albert Polman (2010). "Plasmonics for improved photovoltaic devices". In: *Nature materials* 9.3, p. 205.

Aydin, Koray et al. (2011). "Broadband polarization-independent resonant light absorption using ultrathin plasmonic super absorbers". In: *Nature communications* 2, p. 517.

Baffou, Guillaume et al. (2012). "Thermal imaging of nanostructures by quantitative optical phase analysis". In: *ACS nano* 6.3, pp. 2452–2458.

Baffou, G et al. (2009). "Temperature mapping near plasmonic nanostructures using fluorescence polarization anisotropy". In: *Optics express* 17.5, pp. 3291–3298.

Baily, S. A. and David Emin (2006). "Transport properties of amorphous antimony telluride". In: *Phys. Rev. B* 73 (16), p. 165211. DOI: 10.1103/PhysRevB.73.165211. URL: https://link.aps.org/doi/10.1103/PhysRevB.73.165211.

Bottner, H. et al. (2004). "New thermoelectric components using microsystem technologies". In: *Journal of Microelectromechanical Systems* 13.3, pp. 414–420. ISSN: 1057-7157. DOI: 10.1109/JMEMS.2004.828740.

Briggs, Ryan M., Imogen M. Pryce, and Harry A. Atwater (2010). "Compact silicon photonic waveguide modulator based on the vanadium dioxide metal-insulator phase transition". In: *Opt. Express* 18.11, pp. 11192–11201. DOI: 10.1364/OE.18.011192. URL: http://www.opticsexpress.org/abstract.cfm?URI=oe-18-11-11192.

Buckingham, Michael J (1983). *Noise in electronic devices and systems*. Horwood.

Cai, Xinghan et al. (2014). "Sensitive room-temperature terahertz detection via the photothermoelectric effect in graphene". In: *Nature nanotechnology* 9.10, p. 814.

Carlson, Michael T, Aurangzeb Khan, and Hugh H Richardson (2011). "Local temperature determination of optically excited nanoparticles and nanodots". In: *Nano letters* 11.3, pp. 1061–1069.

Cataldo, Giuseppe et al. (2012). “Infrared dielectric properties of low-stress silicon nitride”. In: *Optics letters* 37.20, pp. 4200–4202.

Chang, Chi-Yang et al. (2007). “Wavelength selective quantum dot infrared photodetector with periodic metal hole arrays”. In: *Applied Physics Letters* 91.16, p. 163107.

Chen, Kai, Ronen Adato, and Hatice Altug (2012). “Dual-band perfect absorber for multispectral plasmon-enhanced infrared spectroscopy”. In: *Acs Nano* 6.9, pp. 7998–8006.

Christ, A et al. (2004). “Optical properties of planar metallic photonic crystal structures: Experiment and theory”. In: *Physical Review B* 70.12, p. 125113.

Coppens, Zachary J et al. (2013a). “Probing and controlling photothermal heat generation in plasmonic nanostructures”. In: *Nano letters* 13.3, pp. 1023–1028.

– (2013b). “Probing and controlling photothermal heat generation in plasmonic nanostructures”. In: *Nano letters* 13.3, pp. 1023–1028.

Datskos, Panos G and Nickolay V Lavrik (2003). “Detectors—Figures of merit”. In: *Encyclopedia of Optical Engineering* 349.

Davis, Kevin L et al. (1993). “Spatially resolved temperature measurements in electrophoresis capillaries by Raman thermometry”. In: *Analytical chemistry* 65.3, pp. 293–298.

Dicke, Robert Henry (1946). “The measurement of thermal radiation at microwave frequencies”. In: *Classics in Radio Astronomy*. Springer, pp. 106–113.

Efetov, Dmitri K et al. (2018). “Fast thermal relaxation in cavity-coupled graphene bolometers with a Johnson noise read-out”. In: *Nature nanotechnology* 13.9, p. 797.

Einstein, Albert (1905). “Investigations on the Theory of the Brownian Movement”. In: *Ann. der Physik*.

Fano, Ugo (1961). “Effects of configuration interaction on intensities and phase shifts”. In: *Physical Review* 124.6, p. 1866.

Fante, R. L. and M. T. McCormack (1988). “Reflection properties of the Salisbury screen”. In: *IEEE Transactions on Antennas and Propagation* 36.10, pp. 1443–1454. ISSN: 0018-926X. DOI: 10.1109/8.8632.

Fischer, Christian and Ioanna Kakoulli (2006). “Multispectral and hyperspectral imaging technologies in conservation: current research and potential applications”. In: *Studies in Conservation* 51.sup1, pp. 3–16.

Fong, Kin Chung and KC Schwab (2012). “Ultrasensitive and wide-bandwidth thermal measurements of graphene at low temperatures”. In: *Physical Review X* 2.3, p. 031006.

Frigo, Matteo and Steven G. Johnson (2005). "The Design and Implementation of FFTW3". In: *Proceedings of the IEEE* 93.2. Special issue on "Program Generation, Optimization, and Platform Adaptation", pp. 216–231.

Ftouni, Hossein et al. (2015). "Thermal conductivity of silicon nitride membranes is not sensitive to stress". In: *Phys. Rev. B* 92 (12), p. 125439. DOI: 10.1103/PhysRevB.92.125439. URL: https://link.aps.org/doi/10.1103/PhysRevB.92.125439.

Gallinet, Benjamin (2012). "Fano Resonances in Plasmonic Nanostructures: Fundamentals, Numerical Modeling and Applications". PhD thesis. Ecole Polytechnique Federale de Lausanne.

Gallinet, Benjamin and Olivier J. F. Martin (2011a). "Influence of Electromagnetic Interactions on the Line Shape of Plasmonic Fano Resonances". In: *ACS Nano* 5.11. PMID: 22026329, pp. 8999–9008. DOI: 10.1021/nn203173r. eprint: https://doi.org/10.1021/nn203173r. URL: https://doi.org/10.1021/nn203173r.

– (2011b). "Relation between near–field and far–field properties of plasmonic Fano resonances". In: *Opt. Express* 19.22, pp. 22167–22175. DOI: 10.1364/OE.19.022167. URL: http://www.opticsexpress.org/abstract.cfm?URI=oe-19-22-22167.

Gallinet, Benjamin and Olivier JF Martin (2011). "Influence of electromagnetic interactions on the line shape of plasmonic Fano resonances". In: *ACS nano* 5.11, pp. 8999–9008.

Garrity, Patrick L (2011). "Spectroscopy of electronic thermal noise as a direct probe of absolute thermoelectric coefficients". In: *Journal of Applied Physics* 109.7, p. 073701.

Gawarikar, A. S., R. P. Shea, and J. J. Talghader (2013a). "High Detectivity Uncooled Thermal Detectors With Resonant Cavity Coupled Absorption in the Long-Wave Infrared". In: *IEEE Transactions on Electron Devices* 60.8, pp. 2586–2591. ISSN: 0018-9383. DOI: 10.1109/TED.2013.2270295.

Gawarikar, Anand S, Ryan P Shea, and Joseph J Talghader (2013b). "High detectivity uncooled thermal detectors with resonant cavity coupled absorption in the long-wave infrared". In: *IEEE Transactions on Electron Devices* 60.8, pp. 2586–2591.

Goetz, Alexander F.H. (2009). "Three decades of hyperspectral remote sensing of the Earth: A personal view". In: *Remote Sensing of Environment* 113. Imaging Spectroscopy Special Issue, S5–S16. ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2007.12.014. URL: http://www.sciencedirect.com/science/article/pii/S003442570900073X.

Golubev, Alexander A et al. (2018). "Plasmonic Heating Plays a Dominant Role in the Plasmon-Induced Photocatalytic Reduction of 4-Nitrobenzenethiol". In: *The Journal of Physical Chemistry C* 122.10, pp. 5657–5663.

Gorbachuk, Nikolai P. and Vladislav R. Sidorko (2004). "Heat Capacity and Enthalpy of Bi2Si3 and Bi2Te3 in the Temperature Range 58-1012 K". In: *Powder Metallurgy and Metal Ceramics* 43.5, pp. 284–290.

Gowen, AA et al. (2007). "Hyperspectral imaging–an emerging process analytical tool for food quality and safety control". In: *Trends in food science & technology* 18.12, pp. 590–598.

Grajower, Meir et al. (2015). "Direct observation of optical near field in nanophotonics devices at the nanoscale using Scanning Thermal Microscopy". In: *Optics express* 23.21, pp. 27763–27775.

Greenwood, N. N. and A. Earnshaw (1984). *Chemistry of the Elements*. Pergamon Press, Inc.

Haberman, R. (2004). *Applied Partial Differential Equations: With Fourier Series and Boundary Value Problems*. Featured Titles for Partial Differential Equations. PEARSON. ISBN: 0-13-065243-1. URL: https://books.google.com/books?id=hGNwLgEACAAJ.

Hammiche, A et al. (1999). "Photothermal FT-IR spectroscopy: A step towards FT-IR microscopy at a resolution better than the diffraction limit". In: *Applied spectroscopy* 53.7, pp. 810–815.

Harman, T. C. et al. (2002). "Quantum Dot Superlattice Thermoelectric Materials and Devices". In: *Science* 297.5590, pp. 2229–2232. ISSN: 0036-8075. DOI: 10.1126/science.1072886. eprint: http://science.sciencemag.org/content/297/5590/2229.full.pdf. URL: http://science.sciencemag.org/content/297/5590/2229.

Harman, TC et al. (2002). "Quantum dot superlattice thermoelectric materials and devices". In: *science* 297.5590, pp. 2229–2232.

Harris, Fredric J (1978). "On the use of windows for harmonic analysis with the discrete Fourier transform". In: *Proceedings of the IEEE* 66.1, pp. 51–83.

Haus, Hermann A and Weiping Huang (1991). "Coupled-mode theory". In: *Proceedings of the IEEE* 79.10, pp. 1505–1518.

Herzog, Joseph B, Mark W Knight, and Douglas Natelson (2014). "Thermoplasmonics: quantifying plasmonic heating in single nanowires". In: *Nano letters* 14.2, pp. 499–503.

Horne, R. A. (1959). "Effect of Oxide Impurities on the Thermoelectric Powers and Electrical Resistivities of Bismuth, Antimony, Tellurium, and Bismuth-Tellurium Alloys". In: *Journal of Applied Physics* 30.3, pp. 393–397. DOI: 10.1063/1.1735175. eprint: https://doi.org/10.1063/1.1735175. URL: https://doi.org/10.1063/1.1735175.

Hsu, Allen L et al. (2015). "Graphene-based thermopile for thermal imaging applications". In: *Nano letters* 15.11, pp. 7211–7216.

Huang, Xiaohua et al. (2008). “Plasmonic photothermal therapy (PPTT) using gold nanoparticles”. In: *Lasers in medical science* 23.3, p. 217.

Imaizumi, Mitsuru et al. (1997). “Effect of grain size and dislocation density on the performance of thin film polycrystalline silicon solar cells”. In: *Journal of applied physics* 81.11, pp. 7635–7640.

Innes, RA and JR Sambles (1985). “Simple thermal detection of surface plasmon-polaritons”. In: *Solid state communications* 56.6, pp. 493–496.

Johnson, John B (1927). “Thermal agitation of electricity in conductors”. In: *Nature* 119.2984, p. 50.

Johnson, John Bertrand (1928). “Thermal agitation of electricity in conductors”. In: *Physical review* 32.1, p. 97.

Johnson, P. B. and R. W. Christy (1972). “Optical Constants of the Noble Metals”. In: *Phys. Rev. B* 6 (12), pp. 4370–4379. DOI: 10.1103/PhysRevB.6.4370. URL: https://link.aps.org/doi/10.1103/PhysRevB.6.4370.

Katzenmeyer, Aaron M et al. (2015). “Mid-infrared spectroscopy beyond the diffraction limit via direct measurement of the photothermal effect”. In: *Nanoscale* 7.42, pp. 17637–17641.

Kay, Art (2012). *Operational Amplifier Noise, 1st ed.* Newnes.

Khorasaninejad, Mohammadreza et al. (2016). “Metalenses at visible wavelengths: Diffraction-limited focusing and subwavelength resolution imaging”. In: *Science* 352.6290, pp. 1190–1194. ISSN: 0036-8075. DOI: 10.1126/science.aaf6644. eprint: http://science.sciencemag.org/content/352/6290/1190.full.pdf. URL: http://science.sciencemag.org/content/352/6290/1190.

Kim, Dong-Ho and Gun-Hwan Lee (2006). “Effect of rapid thermal annealing on thermoelectric properties of bismuth telluride films grown by co-sputtering”. In: *Materials Science and Engineering: B* 131.1, pp. 106–110. ISSN: 0921-5107. DOI: https://doi.org/10.1016/j.mseb.2006.03.034. URL: http://www.sciencedirect.com/science/article/pii/S0921510706002121.

Kim, Kyeongtae et al. (2012). “Ultra-high vacuum scanning thermal microscopy for nanometer resolution quantitative thermometry”. In: *Acs Nano* 6.5, pp. 4248–4257.

Koumoto, K. and T. Mori (2013). *Thermoelectric Nanomaterials: Materials Design and Applications*. Springer.

Kraemer, Daniel et al. (2011). “High-performance flat-panel solar thermoelectric generators with high thermal concentration”. In: *Nature materials* 10.7, p. 532.

Kubo, Rep (1966). “The fluctuation-dissipation theorem”. In: *Reports on progress in physics* 29.1, p. 255.

Kubo, Ryogo, Mario Yokota, and Sadao Nakajima (1957). "Statistical-mechanical theory of irreversible processes. II. Response to thermal disturbance". In: *Journal of the Physical Society of Japan* 12.11, pp. 1203–1211.

Landy, N I et al. (2008). "Perfect metamaterial absorber". In: *Physical review letters* 100.20, p. 207402.

Lenert, Andrej et al. (2014). "A nanophotonic solar thermophotovoltaic device". In: *Nature nanotechnology* 9.2, p. 126.

Lita, Adriana E, Aaron J Miller, and Sae Woo Nam (2008). "Counting near-infrared single-photons with 95% efficiency". In: *Optics express* 16.5, pp. 3032–3040.

Liu, Na et al. (2010a). "Infrared Perfect Absorber and Its Application As Plasmonic Sensor". In: *Nano Letters* 10.7. PMID: 20560590, pp. 2342–2348. DOI: 10.1021/nl9041033. eprint: https://doi.org/10.1021/nl9041033. URL: https://doi.org/10.1021/nl9041033.

– (2010b). "Infrared perfect absorber and its application as plasmonic sensor". In: *Nano letters* 10.7, pp. 2342–2348.

Lu, Guolan and Baowei Fei (2014). "Medical hyperspectral imaging: a review". In: *Journal of biomedical optics* 19.1, p. 010901.

Lumerical. *FDTD Solutions*. URL: http://www.lumerical.com/tcad-products/fdtd/.

Mauser, Kelly W et al. (2017). "Resonant thermoelectric nanophotonics". In: *Nature nanotechnology* 12.8, p. 770.

Moharam, MG et al. (1995). "Formulation for stable and efficient implementation of the rigorous coupled-wave analysis of binary gratings". In: *JOSA a* 12.5, pp. 1068–1076.

Mokkapati, Sudha et al. (2015). "Optical design of nanowire absorbers for wavelength selective photodetectors". In: *Scientific reports* 5, p. 15339.

Moore, JP and RS Graves (1973). "Absolute Seebeck coefficient of platinum from 80 to 340 K and the thermal and electrical conductivities of lead from 80 to 400 K". In: *Journal of Applied Physics* 44.3, pp. 1174–1178.

Motchenbacher, C. D. and J. A. Connelly (1992). *Low-Noise Electronic System Design*. Wiley.

Njoki, Peter N et al. (2007). "Size correlation of optical and spectroscopic properties for gold nanoparticles". In: *The Journal of Physical Chemistry C* 111.40, pp. 14664–14669.

Nolas, G. S., J. Sharp, and H. J. Goldsmid (2001). *Thermoelectrics: Basic Principles and New Materials Developments*. Springer.

Nyquist, Harry (1928). "Thermal agitation of electric charge in conductors". In: *Physical review* 32.1, p. 110.

Palik, E. (1997a). *Handbook of Optical Constants of Solids*. Academic Press.

– (1997b). *Handbook of Optical Constants of Solids*. Academic Press.

Pinisetty, D. et al. (2011). "Fabrication and characterization of electrodeposited antimony telluride crystalline nanowires and nanotubes". In: *J. Mater. Chem.* 21 (12), pp. 4098–4107. DOI: 10.1039/C0JM01969K. URL: http://dx.doi.org/10.1039/C0JM01969K.

Poudel, Bed, Qing Hao, Yi Ma, Yucheng Lan, Austin Minnich, Bo Yu, Xiao Yan, Dezhi Wang, Andrew Muto, Daryoosh Vashaee, et al. (2008a). "High-thermoelectric performance of nanostructured bismuth antimony telluride bulk alloys". In: *Science* 320.5876, pp. 634–638.

Poudel, Bed, Qing Hao, Yi Ma, Yucheng Lan, Austin Minnich, Bo Yu, Xiao Yan, Dezhi Wang, Andrew Muto, Daryoosh Vashaee, et al. (2008b). "High-Thermoelectric Performance of Nanostructured Bismuth Antimony Telluride Bulk Alloys". In: *Science* 320.5876, pp. 634–638. ISSN: 0036-8075. DOI: 10.1126/science.1156446. eprint: http://science.sciencemag.org/content/320/5876/634.full.pdf. URL: http://science.sciencemag.org/content/320/5876/634.

Qu, Jifeng et al. (2013). "Flat frequency response in the electronic measurement of Boltzmann's constant". In: *IEEE Transactions on Instrumentation and Measurement* 62.6, pp. 1518–1523.

Rajagopalan, N.S. and S.K. Ghosh (1963). "Measurements of some electrical and thermoelectrical properties of antimony telluride films". In: *Physica* 29.3, pp. 234–242. ISSN: 0031-8914. DOI: https://doi.org/10.1016/S0031-8914(63)80282-7. URL: http://www.sciencedirect.com/science/article/pii/S0031891463802827.

Rashid, Mohammad Mamunur, Kyung Ho Cho, and Gwiy-Sang Chung (2013). "Rapid thermal annealing effects on the microstructure and the thermoelectric properties of electrodeposited Bi2Te3 film". In: *Applied Surface Science* 279, pp. 23–30. ISSN: 0169-4332. DOI: https://doi.org/10.1016/j.apsusc.2013.03.112. URL: http://www.sciencedirect.com/science/article/pii/S0169433213005977.

Rosencwaig, Allan et al. (1985). "Detection of thermal waves through optical reflectance". In: *Applied Physics Letters* 46.11, pp. 1013–1015. DOI: 10.1063/1.95794. eprint: https://doi.org/10.1063/1.95794. URL: https://doi.org/10.1063/1.95794.

Rowe, D. M. and C. M. Bhandari (1995). *CRC Handbook of Thermoelectrics*. CRC.

Russer, Johannes A et al. (2015). "A nanostructured long-wave infrared range thermocouple detector". In: *IEEE Transactions on Terahertz Science and Technology* 5.3, pp. 335–343.

Salisbury, Winfield W (1952). *Absorbent body for electromagnetic waves*. US Patent 2,599,944.

Schoenlein, RW et al. (1987). "Femtosecond studies of nonequilibrium electronic processes in metals". In: *Physical Review Letters* 58.16, p. 1680.

Schwab, Keith et al. (2000). "Measurement of the quantum of thermal conductance". In: *Nature* 404.6781, p. 974.

Shi, Weidong et al. (2008). "Hydrothermal Synthesis and Thermoelectric Transport Properties of Impurity-Free Antimony Telluride Hexagonal Nanoplates". In: *Advanced Materials* 20.10, pp. 1892–1897. DOI: 10.1002/adma.200702003. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/adma.200702003. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.200702003.

Silva, Luciana W. da, Massoud Kaviany, and Ctirad Uher (2005). "Thermoelectric performance of films in the bismuth-tellurium and antimony-tellurium systems". In: *Journal of Applied Physics* 97.11, p. 114903. DOI: 10.1063/1.1914948. eprint: https://doi.org/10.1063/1.1914948. URL: https://doi.org/10.1063/1.1914948.

Singh, Rajeev and Ali Shakouri (2009). "Thermostat for high temperature and transient characterization of thin film thermoelectric materials". In: *Review of Scientific Instruments* 80.2, p. 025101. DOI: 10.1063/1.3072603. eprint: https://doi.org/10.1063/1.3072603. URL: https://doi.org/10.1063/1.3072603.

Skoplaki, Elisa and John A Palyvos (2009). "On the temperature dependence of photovoltaic module electrical performance: A review of efficiency/power correlations". In: *Solar energy* 83.5, pp. 614–624.

Stoica, Petre, Randolph L Moses, et al. (2005). "Spectral analysis of signals". In:

Sung, Ji Ho et al. (2014). "Atomic Layer-by-Layer Thermoelectric Conversion in Topological Insulator Bismuth/Antimony Tellurides". In: *Nano Letters* 14.7. PMID: 24937706, pp. 4030–4035. DOI: 10.1021/nl501468k. eprint: https://doi.org/10.1021/nl501468k. URL: https://doi.org/10.1021/nl501468k.

Szakmany, Gergo P et al. (2015). "Novel nanoscale single-metal polarization-sensitive infrared detectors". In: *IEEE Transactions on Nanotechnology* 14.2, pp. 379–383.

Takashiri, M. et al. (2008). "Effect of grain size on thermoelectric properties of n-type nanocrystalline bismuth-telluride based thin films". In: *Journal of Applied Physics* 104.8, p. 084302. DOI: 10.1063/1.2990774. eprint: https://doi.org/10.1063/1.2990774. URL: https://doi.org/10.1063/1.2990774.

Treble, F.C (1962). “The effects of radiation damage in solar cells”. In: *Microelectronics Reliability* 1.4, pp. 299–309. ISSN: 0026-2714. DOI: https://doi.org/10.1016/0026-2714(62)90034-3. URL: http://www.sciencedirect.com/science/article/pii/0026271462900343.

Tsiatmas, Anagnostis et al. (2013). “Optical generation of intense ultrashort magnetic pulses at the nanoscale”. In: *New Journal of Physics* 15.11, p. 113035.

Vining, Cronin B (2009). “An inconvenient truth about thermoelectrics”. In: *Nature materials* 8.2, p. 83.

Völklein, F and A Wiegand (1990). “High sensitivity and detectivity radiation thermopiles made by multi-layer technology”. In: *Sensors and Actuators A: Physical* 24.1, pp. 1–4.

Völklein, F. and A. Wiegand (1990). “High sensitivity and detectivity radiation thermopiles made by multi-layer technology”. In: *Sensors and Actuators A: Physical* 24.1, pp. 1–4. ISSN: 0924-4247. DOI: https://doi.org/10.1016/0924-4247(90)80040-C. URL: http://www.sciencedirect.com/science/article/pii/092442479080040C.

Wang, H. and K. J. Leonard (2017). “Effect of high fluence neutron irradiation on transport properties of thermoelectrics”. In: *Applied Physics Letters* 111.4, p. 043901. DOI: 10.1063/1.4990714. eprint: https://doi.org/10.1063/1.4990714. URL: https://doi.org/10.1063/1.4990714.

Wang, W. and Z. M. Wang (2014). *Nanoscale Thermoelectrics*. Springer.

Ward, Daniel R et al. (2011). “Vibrational and electronic heating in nanoscale junctions”. In: *Nature nanotechnology* 6.1, p. 33.

Weeber, J-C et al. (2011). “Thermo-electric detection of waveguided surface plasmon propagation”. In: *Applied Physics Letters* 99.3, p. 031113.

Welch, Peter (1967). “The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms”. In: *IEEE Transactions on audio and electroacoustics* 15.2, pp. 70–73.

White, DR et al. (1996). “The status of Johnson noise thermometry”. In: *Metrologia* 33.4, p. 325.

Wipiejewski, T, K Panzlaff, and Karl Joachim Ebeling (1992). “Resonant wavelength selective photodetectors”. In: *Microelectronic Engineering* 19.1-4, pp. 223–226.

Xu, Xiaodong et al. (2009). “Photo-thermoelectric effect at a graphene interface junction”. In: *Nano letters* 10.2, pp. 562–566.

Yamane, Tsuneyuki et al. (2002). “Measurement of thermal conductivity of silicon dioxide thin films using a 3 method”. In: *Journal of Applied Physics* 91.12, pp. 9772–9776. DOI: 10.1063/1.1481958. eprint: https://aip.scitation.org/doi/pdf/10.1063/1.1481958. URL: https://aip.scitation.org/doi/abs/10.1063/1.1481958.

Yang, Ronggui, Arvind Narayanaswamy, and Gang Chen (2005). “Surface-plasmon coupled nonequilibrium thermoelectric refrigerators and power generators”. In: *Journal of Computational and Theoretical Nanoscience* 2.1, pp. 75–87.

Yeng, Yi Xiang et al. (2012). “Enabling high-temperature nanophotonics for energy applications”. In: *Proceedings of the National Academy of Sciences* 109.7, pp. 2280–2285.

Zhang, Xiao and Li-Dong Zhao (2015). “Thermoelectric materials: Energy conversion between heat and electricity”. In: *Journal of Materiomics* 1.2, pp. 92–105. ISSN: 2352-8478. DOI: https://doi.org/10.1016/j.jmat.2015.01.001. URL: http://www.sciencedirect.com/science/article/pii/S2352847815000258.

Zou, Helin, D.M. Rowe, and S.G.K. Williams (2002). “Peltier effect in a co-evaporated Sb2Te3(P)-Bi2Te3(N) thin film thermocouple”. In: *Thin Solid Films* 408.1, pp. 270–274. ISSN: 0040-6090. DOI: https://doi.org/10.1016/S0040-6090(02)00077-9. URL: http://www.sciencedirect.com/science/article/pii/S0040609002000779.

Appendix A

FFTW C PROGRAM FOR DATA ANALYSIS

From www.fftw.org, *"FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST)."*. FFTW (Frigo and S. Johnson, 2005) stands for "Fastest Fourier Transform in the West", and is free. We found Matlab to be impossibly slow at analyzing data, and Python to take several hours with multiprocessing to analyze data. We wrote this data analysis program in C using the FFTW library, and it has cut down the data analysis time from 3-5 hours in Python to approximately 20 minutes in C. One of the components that was severely slowing the Python program was simply opening the large binary files the data was stored in. C takes a fraction of a second to access the data. While Python has built-in Welch functions, we had to write our own in C, which is detailed in an earlier chapter of this thesis.

The FFTW program can be installed on most Linux computers by using the general installation procedure. On CentOS, for example, type the following into a terminal:

sudo yum install fftw

followed by

sudo yum install fftw-devel

If using a Windows operating system, VirtualBox (with RedHat and CentOS) works with fftw, as does Linux Subsystem for Windows, which is available in the Windows Store. The Linux Subsystem is easier for accessing files shared between the Windows and Linux system, but the VirtualBox is much more forgiving when trying new techniques, as you can delete the Virtual Machine and make a new one if you generate serious problems, which you can't do with Linux Subsystem. In our application, we required a Windows machine to take data with the NI USB 6366, as National Instruments only created a USB driver for this DAQ for Windows machines, and not Linux, as of this writing. The PXIe version of the 6366 has a Linux driver as well as a Windows driver, but is less portable than the USB version, and cannot as easily be transferred between computers.

We used a Python program with wrappers for C to communicate with and collect data

from the NI USB6366, using the Python package PyDAQmx, which allows Python to interact with the DAQmx driver for certain National Instruments products. This data was saved in binary files.

If installed the standard way, using a gcc compiler, the compile command is

gcc -Wall fftw_data_analysis_program.c -I$HOME/usr/include -L$HOME/usr/lib -lfftw3 -lm -o fftw_executable_program

The C program we wrote took several arguments at the command line, a path to a directory with the data to analyze, and the extension for the analyzed data text file, among other inputs. We changed the .c program and recompiled if we were changing the order of resistors in the circuit, or were using chopped data or unchopped data (baseline or control). To run data analysis in parallel, threading can be used with FFTW, though we chose to use bash scripting. An example script is below:

```
#!/bin/bash

for num in {0..10}
do
fftw_executable path_to_data _analyzed_data_$num other_args &
done
wait
```

GNUplot, a free Linux plotting software, was also installed for examining the data to determine the transition region truncation. The following is the full data analysis program. We apologize for un-optimized coding, but we appreciate that C is fast enough to compensate for poor form. The code explained in detail in a previous chapter.

```
/* Compile with

gcc -Wall program_name.c -I$HOME/usr/include -L$HOME/usr/lib -lfftw3
    ↪ -lm -o analysis_excecutable

Run with (for example)
* /analysis_excecutable /path_to_data_to_analyze
    ↪ _suffix_for_analyzed_file 1 1 1 1 right_freq_limit
    ↪ left_freq_limit 3 R1_freq R2_freq

```

```
7  */
8
9  //complex.h needs to be above fftw3.h for this program to work, as
       ↪ complex.h complex number formatting is used instead of fftw3.h
       ↪ . See fftw documentation for details.
10 #include <complex.h>
11 #include <fftw3.h>
12 #include <math.h>
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <sys/stat.h>
16 #include <dirent.h>
17 #include <string.h>
18
19 #define PI 3.14159265
20 #define KB 1.38064852E-23
21
22
23
24 //Windowing function - we chose blackmanharris - for noise
       ↪ measurements of very large data sets, the window choice does
       ↪ not (and should not) have an effect.
25 double blackmanharris(long n,long segment_length);
26
27 //Linear interpolation function
28 double linear_interpy(double x_new, double *x_base, double *y_base,
       ↪ double max_freqq);
29
30 //Finding complex impedence at a given frequency and source
       ↪ resistance
31 double find_magz(double freq_val, double rss);
32
33 //Calculate the noise from external sources.
34 double external_noise_calc(double gain_data, double daq, double r4,
       ↪ double temp, double in, double r3, double gainz2, double en,
       ↪ double zout,double r2, double r1, double gainz1, double rs );
35
36 //Plots index versus ydata in gnuplot - must have gnuplot installed
```

```
37 void plotplot(long data_len, fftw_complex *ydata);
38
39 //Plots index versus 2 ydatas in gnuplot
40 void plotmultiplot(long data_len,double *ydata1, double *ydata2);
41
42 //Plots loglog in gnuplot (doesn't yet read in x data)
43 void plotlogplot(long data_len,double *ydata);
44
45 //Plots loglog in gnuplot the fft data for TEV
46 void plotlogplotfft(long data_len,long nyq,double *ydata);
47
48 //Plot 8 lines in loglog (for checking noise data components)
49 void plotlogplot8(long data_len,double *ydata1,double *ydata2,double
       ↪ *ydata3,double *ydata4,double *ydata5,double *ydata6,double *
       ↪ ydata7,double *ydata8);
50
51 //Plot many things
52 void plotlogplotmany(int num_rows,long data_len,double *total_meas,
       ↪ double **cirq_noise);
53
54 //Make 2D array (not for use with fftw arrays, not supported)
55 double** array2D(int xdim, int ydim);
56
57 //function to find amplitude of frequency value. fft_length is #
       ↪ samples/2 +1
58 void FindAmpAtFreq(double freq_of_interest, double fft_length, double
       ↪  nyquist_freq, double *fft_to_search, double *max_value,
       ↪ double *max_index);
59
60  int main(int argc, char **argv){
61
62        if( argc != 12){
63               printf("Exactly 11 arguments needed, path to data
                   ↪ directory, save extension, 4 numbers of
                   ↪ resistance in ohms, and right cut (number) and
                   ↪ left cut (number), chopper freq, and CH2 R
                   ↪ frequency and CH3 R frequency (ex. /home/
                   ↪ kamauser/C_programs/20190308_heater_cals/
```

```
                ↪ Data_binary_20190308_cirq_1_2/ and cirq1_2 and
                ↪ 3300 4700 1500 2200 and 12000 2000 and 43 and
                ↪ 21630 18387).\n");
64              exit(-1);
65      }
66
67  //SET DATA ANALYSIS FORMAT (WITH/WITHOUT GAIN, ETC...)
68  //
    ↪ ********************************************************************************
    ↪
69
70      //Is it a flat spectrum (as in, unchopped, so a control or
            ↪ temperature calibration run)? 1 = yes, 0 = is chopped
71      int is_flat = 0;
72
73  //If is_flat = 0, set this equal to 0 (or problems may arise).
        ↪ Otherwise, if is_flat=1, this will use the TEV extractor to
        ↪  get the R value if you feed the chopper frequency as the R
        ↪  frequency. The answer is NOT multiplied by 2pi/4 (pi/4 for
        ↪  square wave correction, 2 for Vpp) as it is for is_flat
        ↪ =0.
74  int getRconst = 0;
75
76      //data path
77      char data_path[500];//
78
79      //Do you want raw data or data with gain versus freq corrected
            ↪  and circuit component noise contributions subtracted?
80      int raw_data = 0; //1 = raw data, 0 = data with above
            ↪ calculations
81
82      //If raw_data = 0, do you want to use theoretical gain (r2/r1
            ↪ +1)*(r4/r3+1) or measured gain?
83      int theo_gain = 1;//1 means using theoretical gain on all
            ↪ frequencies. If raw_data = 1, make theo_gain = 0.
84
85  //If tev_gain_meas = 1, then a separate gain file is used during
        ↪ tev analysis which overrides the raw_data and theo_gain for
```

```
      ↪  tev voltages. This is useful if you want theoretical gain
      ↪ at high frequencies, but can't use it at low frequencies
      ↪ due to lowpass filtering, or if there is separate frequency
      ↪ -related attenuation analysis needed. If tev_gain_meas = 0,
      ↪  the same gain file is used for tev analysis as for noise
      ↪ analysis. (so set to 0 if is_flat=1 and getRconst=1 if the
      ↪ tev gain file doesn't go to high enough frequencies).
86    int tev_gain_meas = 1;
87
88        //save data path
89        char filename_nsd_on[500];// =
90        char filename_nsd_off[500];// =
91        char filename_tev[500];// =
92        char filename_R_on[500];
93        char filename_R_off[500];
94
95
96  //MAKE SAVE FILE NAMES
97  //
    ↪ ********************************************************************************
    ↪
98        //Read in the arguments for the file path and data save
              ↪ extension.
99        char data_ext[100];
100       sscanf(argv[1], "%s", data_path);
101       sscanf(argv[2], "%s", data_ext);
102       snprintf(filename_nsd_on,500,"/path/NSD_on_data%s.txt",
              ↪ data_ext);
103       snprintf(filename_nsd_off,500,"/path/NSD_off_data%s.txt",
              ↪ data_ext);
104       snprintf(filename_tev,500,"/path/NSD_tev%s.txt",data_ext);
105       snprintf(filename_R_on,500,"/path/R_on%s.txt",data_ext);
106       snprintf(filename_R_off,500,"/path/R_off%s.txt",data_ext);
107       printf("path to data files: %s",filename_nsd_off);
108
109 //GAIN FILE LOCATIONS
110 //
    ↪ ********************************************************************************
```

```
    ↪
111
112         char gain1[500];
113         char gain2[500];
114         char gain3[500];
115         char gain4[500];
116     if(raw_data == 1){//Don't change these - these give a gain of 1 -
            ↪ and will give raw data if that is desired
117       snprintf(gain1,500,"/path/gainversusfreq_201811/ch1_gain_1.txt")
              ↪ ;
118       snprintf(gain2,500,"/path/C_programs/gainversusfreq_201811/
              ↪ ch2_gain_1.txt");
119       snprintf(gain3,500,"/path/C_programs/gainversusfreq_201811/
              ↪ ch3_gain_1.txt");
120       snprintf(gain4,500,"/path/C_programs/gainversusfreq_201811/
              ↪ ch4_gain_1.txt");
121     }
122     else if(raw_data ==0){//change these to the path where you have
            ↪ gain versus frequency data
123       snprintf(gain1,500,"/path/
              ↪ lockin_data_20190308_opamp4_3p3k_4p76mV_finest_ch1.csv");
124       snprintf(gain2,500,"/path/
              ↪ lockin_data_20190308_opamp4_3p3k_4p76mV_finest_ch2.csv");
125       snprintf(gain3,500,"/path/
              ↪ lockin_data_20190308_opamp4_3p3k_4p76mV_finest_ch3.csv");
126       snprintf(gain4,500,"/path/
              ↪ lockin_data_20190308_opamp4_3p3k_4p76mV_finest_ch4.csv");
127     }
128     else{
129       printf("ERROR: raw_data must be either 0 or 1\n");
130       exit(-1);
131     }
132
133     //TEV gain file locations - theoretical opamp gain is not valid at
            ↪  low frequencies because of low-pass filtering in circuit
134         char gain1_tev[500];//These must be declared outside of if
                ↪ function
135         char gain2_tev[500];
```

```
136         char gain3_tev[500];
137         char gain4_tev[500];
138 
139     if(tev_gain_meas == 1){
140 
141       snprintf(gain1_tev,500,"/path/
              ↪ lockin_data_20190308_opamp4_3p3k_4p76mV_finest_ch1.csv");
142       snprintf(gain2_tev,500,"/path/
              ↪ lockin_data_20190308_opamp4_3p3k_4p76mV_finest_ch2.csv");
143       snprintf(gain3_tev,500,"/path/
              ↪ lockin_data_20190308_opamp4_3p3k_4p76mV_finest_ch3.csv");
144       snprintf(gain4_tev,500,"/path/
              ↪ lockin_data_20190308_opamp4_3p3k_4p76mV_finest_ch4.csv");
145     }
146 
147 
148 //OTHER VARIABLES
149 //
      ↪ ********************************************************************************
      ↪ 
150 
151 
152         //Welch variables for later - we assume a 50% overlap
153         double segment_len = 2097152; //Length of welch segmentprint -
              ↪  should be faster if a power of 2 (2^21 = 2097152, 2^15
              ↪  = 32768, 2^16 = 65536, 2^25 = 33554432, 2^26 =
              ↪ 67108864) This actually works. Don't make larger than
              ↪ ~1/5 of data size. Smaller is better. Because split
              ↪ into on/off and ~1/2 of data thrown out, you could end
              ↪ up with segment_len>fft size
154         double segment_len_4fft = 100000000;//For taking an FFT to get
              ↪  thermoelectric voltage (we don't have to FFT all the
              ↪ data)
155 
156         double samp_rate = 2000000; //sample rate during measurement
157 
158         double chop_rate = atof(argv[9]);//43.0; //chopper frequency,
              ↪ used to guide the TEV peak-finder in the fft, will
```

```
            ↪ search about +/-5 Hz for the peak
159
160         //Frequency to average between to get the average PSD
161         double freq_min = 100000; //White noise frequency minimum
162         double freq_max = 900000; //White noise frequency maximum
163
164         //This is for cutting the transition region, in units of index
                ↪ . The data should be plotted and this determined.
165         long right_cut = atol(argv[7]);//12000; //Number of values to
                ↪ the right of the transition cutoff to remove
166         long left_cut = atol(argv[8]);//2000; //Number of values to
                ↪ the left of the transition cutoff to remove
167
168         //Length of segment for determining resistance (should be
                ↪ approximately the size of the region analyzed for on/
                ↪ off
169         double r_segment_len = 0.5*(1.0/((double)chop_rate))*samp_rate
                ↪  - right_cut - left_cut;
170
171         //Frequencies used for measuring resistance
172         double ch2_r_freq = atof(argv[10]);
173         double ch3_r_freq = atof(argv[11]);
174
175
176 //PLOTTING FOR ERROR CHECKING
177 //
    ↪ ********************************************************************************
    ↪
178         int plot_data = 0; //Set 1 to see data when setting right_cut
                ↪ and left_cut
179         long ind2plot = 50000; //Number of values to plot
180         int plot_psd_cumul_array = 0;//set equal to one to see the psd
                ↪  cumulative array
181         int plot_separate_source = 0;//1 means each contribution to
                ↪ noise will be plotted (error checking)
182         int plot_errchk = 0;//for exhaustive error checking
183         int plot_fft = 0;
184       int plot_fft_R = 0;
```

```
185
186
187 //NUMBER OF CHANNELS AND SCANS
188 //
    ↪ ********************************************************************************
    ↪
189         int num_scans = 5;//Number of runs you wish to analyze within
                ↪ the data folder
190         int num_chans = 4;//NSD channels, not chopper channels
191
192
193 //RESISTANCE OF EACH CHANNEL AND NOISE CALCULATIONS
194 //
    ↪ ********************************************************************************
    ↪
195
196         double r_of_chans[4];//in order from ch1, ch2, ch3, ch4
197         sscanf(argv[3], "%lf", &r_of_chans[0]);//Put resistances from
                ↪ command line argumetns into r_of_chans array
198         sscanf(argv[4], "%lf", &r_of_chans[1]);
199         sscanf(argv[5], "%lf", &r_of_chans[2]);
200         sscanf(argv[6], "%lf", &r_of_chans[3]);
201
202
203         double r1 = 33.2;//smaller gain resistor first stage
204         double r2 = 300;//larger gain resistor first stage
205         double r3 = 33.2;//smaller gain resistor second stage
206         double r4 = 300;//larger gain resistor second stage
207         double zout = 20;//output impedance of first stage opamp (
                ↪ seriously negligible)
208         double temp = 298;//temperature of gain resistors
209         double en = 3.8E-9;//input voltage noise of amplifier (in V/
                ↪ sqrt(Hz))
210         double in = 2.2E-15;//current noise of amplifier (in A/sqrt(Hz
                ↪ ))
211         double daq = 38e-9;//voltage noise of DAQ (in V/sqrt(Hz))
212
213         //Calculations for extraction of noise from opamp, don't
```

```
            ↪ change these unless configuration changes
214         double gainz1, gainz2;
215         gainz1 = r2/r1 + 1;//Gain of first stage amplifier
216         gainz2 = r4/r3 + 1;//Gain of second stage amplifier
217
218
219 //GAIN CORRECTION INTERPOLATIONS
220 //
    ↪ ********************************************************************************
    ↪
221         //Read in txtfiles for gain corrections
222         printf("reading in gain corrections\n");
223         FILE *ch1_gptr,*ch2_gptr,*ch3_gptr,*ch4_gptr;
224         long gain_file_size = 0;
225         double * gain_file_freq_data;
226         double *gain_ch1_data,*gain_ch2_data,*gain_ch3_data,*
            ↪ gain_ch4_data;
227         ch1_gptr = fopen(gain1,"r");
228         ch2_gptr = fopen(gain2,"r");
229         ch3_gptr = fopen(gain3,"r");
230         ch4_gptr = fopen(gain4,"r");
231         if(ch1_gptr == NULL){
232                printf("error reading gain files\n");
233                exit(0);
234         }
235         if(ch2_gptr == NULL){
236                printf("error reading gain files\n");
237                exit(0);
238         }
239         if(ch3_gptr == NULL){
240                printf("error reading gain files\n");
241                exit(0);
242         }
243         if(ch4_gptr == NULL){
244                printf("error reading gain files\n");
245                exit(0);
246         }
247         //find the number of lines
```

```
248         printf("finding number of lines in gain file\n");
249         char line[128];
250         while(fgets(line,sizeof(line),ch1_gptr)!=NULL){
251                 gain_file_size++;
252         }
253
254         printf("size of gain_file_size: %ld\n",gain_file_size);
255         fseek(ch1_gptr,0,SEEK_SET);
256
257     //malloc space for gain arrays
258         gain_file_freq_data = (double *)malloc(sizeof(double)*
                ↪ gain_file_size);
259         gain_ch1_data = (double *)malloc(sizeof(double)*gain_file_size
                ↪ );
260         gain_ch2_data = (double *)malloc(sizeof(double)*gain_file_size
                ↪ );
261         gain_ch3_data = (double *)malloc(sizeof(double)*gain_file_size
                ↪ );
262         gain_ch4_data = (double *)malloc(sizeof(double)*gain_file_size
                ↪ );
263
264         //put the data into the arrays
265         int hm=0;
266         char str1[100];
267         char str2[100];
268         char str3[100];
269         char str4[100];
270         for(hm=0;hm<gain_file_size;hm++){
271                 fgets(str1,sizeof(str1),ch1_gptr);
272                 fgets(str2,sizeof(str2),ch2_gptr);
273                 fgets(str3,sizeof(str3),ch3_gptr);
274                 fgets(str4,sizeof(str4),ch4_gptr);
275                 sscanf(str1,"%le,%le",&gain_file_freq_data[hm],&
                        ↪ gain_ch1_data[hm]);
276                 sscanf(str2,"%*e,%le",&gain_ch2_data[hm]);
277                 sscanf(str3,"%*e,%le",&gain_ch3_data[hm]);
278                 sscanf(str4,"%*e,%le",&gain_ch4_data[hm]);
279         }
```

```
280
281
282         int testint;
283         for(testint=0;testint<3;testint++){
284                 printf("gain_file_freq_data = %e\n",gain_file_freq_data
                        ↪ [testint]);
285         }
286
287         fclose(ch1_gptr);
288         fclose(ch2_gptr);
289         fclose(ch3_gptr);
290         fclose(ch4_gptr);
291
292         printf("read in gain corrections successfully\n");
293
294         printf("interpolating gain corrections for frequencies\n");
295         double *gain_ch1_data_interp;
296         double *gain_ch2_data_interp;
297         double *gain_ch3_data_interp;
298         double *gain_ch4_data_interp;
299         gain_ch1_data_interp = (double *)malloc(sizeof(double)*(
            ↪ segment_len/2+1));
300         gain_ch2_data_interp = (double *)malloc(sizeof(double)*(
            ↪ segment_len/2+1));
301         gain_ch3_data_interp = (double *)malloc(sizeof(double)*(
            ↪ segment_len/2+1));
302         gain_ch4_data_interp = (double *)malloc(sizeof(double)*(
            ↪ segment_len/2+1));
303         int freq_cntr2;
304         double freqq2;
305         if(theo_gain==0){
306                 for(freq_cntr2 = 0;freq_cntr2<(segment_len/2 + 1);
                        ↪ freq_cntr2++){
307                         freqq2 = freq_cntr2*((samp_rate/2)/(segment_len
                                ↪ /2 +1));//frequency needed
308                         gain_ch1_data_interp[freq_cntr2] =
                                ↪ linear_interpy(freqq2,gain_file_freq_data
                                ↪ ,gain_ch1_data,samp_rate/2.0);
```

```
309                     gain_ch2_data_interp[freq_cntr2] =
                            ↪ linear_interpy(freqq2,gain_file_freq_data
                            ↪ ,gain_ch2_data,samp_rate/2.0);
310                     gain_ch3_data_interp[freq_cntr2] =
                            ↪ linear_interpy(freqq2,gain_file_freq_data
                            ↪ ,gain_ch3_data,samp_rate/2.0);
311                     gain_ch4_data_interp[freq_cntr2] =
                            ↪ linear_interpy(freqq2,gain_file_freq_data
                            ↪ ,gain_ch4_data,samp_rate/2.0);
312             }
313     }
314     else if(theo_gain==1){//using theoretical gain
315             for(freq_cntr2 = 0;freq_cntr2<(segment_len/2 + 1);
                    ↪ freq_cntr2++){
316                     freqq2 = freq_cntr2*((samp_rate/2)/(segment_len
                            ↪ /2 +1));//frequency needed
317                     gain_ch1_data_interp[freq_cntr2] = gainz1*gainz2
                            ↪ ;
318                     gain_ch2_data_interp[freq_cntr2] = gainz1*gainz2
                            ↪ ;
319                     gain_ch3_data_interp[freq_cntr2] = gainz1*gainz2
                            ↪ ;
320                     gain_ch4_data_interp[freq_cntr2] = gainz1*gainz2
                            ↪ ;
321             }
322     }
323   else{
324     printf("ERROR: theo_gain must be either 0 or 1\n");
325             exit(-1);
326   }
327     printf("interpolation done\n");
328
329     for(testint=0;testint<10;testint++){
330             printf("gain data ch1 line %d is: %f\n",testint,
                    ↪ gain_ch1_data_interp[testint]);
331     }
332
333 //TEV GAIN CORRECTION INTERPOLATIONS (IF tev_gain_meas == 1)
```

```
334 //
    ↪ ********************************************************************************
    ↪
335         double * gain_file_freq_data_tev;//must be declared outside of
                ↪  if statement
336         double *gain_ch1_data_tev,*gain_ch2_data_tev,*
                ↪ gain_ch3_data_tev,*gain_ch4_data_tev;
337     if(tev_gain_meas == 1){
338           //Read in txtfiles for gain corrections
339          printf("reading in gain corrections\n");
340          FILE *ch1_gptr_tev,*ch2_gptr_tev,*ch3_gptr_tev,*ch4_gptr_tev;
341          long gain_file_size_tev = 0;
342
343          ch1_gptr_tev = fopen(gain1_tev,"r");
344          ch2_gptr_tev = fopen(gain2_tev,"r");
345          ch3_gptr_tev = fopen(gain3_tev,"r");
346          ch4_gptr_tev = fopen(gain4_tev,"r");
347          if(ch1_gptr_tev == NULL){
348                 printf("error reading tev gain files\n");
349                 exit(0);
350          }
351          if(ch2_gptr_tev == NULL){
352                 printf("error reading tev gain files\n");
353                 exit(0);
354          }
355          if(ch3_gptr_tev == NULL){
356                 printf("error reading tev gain files\n");
357                 exit(0);
358          }
359          if(ch4_gptr_tev == NULL){
360                 printf("error reading tev gain files\n");
361                 exit(0);
362          }
363          //find the number of lines
364          printf("finding number of lines in gain file\n");
365          char line[128];
366          while(fgets(line,sizeof(line),ch1_gptr_tev)!=NULL){
367                 gain_file_size_tev++;
```

```
368         }
369
370         printf("size of gain_file_size: %ld\n",gain_file_size_tev);
371         fseek(ch1_gptr_tev,0,SEEK_SET);
372
373       //malloc space for gain arrays
374         gain_file_freq_data_tev = (double *)malloc(sizeof(double)*
              ↪ gain_file_size_tev);
375         gain_ch1_data_tev = (double *)malloc(sizeof(double)*
              ↪ gain_file_size_tev);
376         gain_ch2_data_tev = (double *)malloc(sizeof(double)*
              ↪ gain_file_size_tev);
377         gain_ch3_data_tev = (double *)malloc(sizeof(double)*
              ↪ gain_file_size_tev);
378         gain_ch4_data_tev = (double *)malloc(sizeof(double)*
              ↪ gain_file_size_tev);
379
380         //put the data into the arrays
381         int hm=0;
382         char str1[100];
383         char str2[100];
384         char str3[100];
385         char str4[100];
386         for(hm=0;hm<gain_file_size_tev;hm++){
387                fgets(str1,sizeof(str1),ch1_gptr_tev);
388                fgets(str2,sizeof(str2),ch2_gptr_tev);
389                fgets(str3,sizeof(str3),ch3_gptr_tev);
390                fgets(str4,sizeof(str4),ch4_gptr_tev);
391                sscanf(str1,"%le,%le",&gain_file_freq_data_tev[hm],&
                     ↪ gain_ch1_data_tev[hm]);
392                sscanf(str2,"%*e,%le",&gain_ch2_data_tev[hm]);
393                sscanf(str3,"%*e,%le",&gain_ch3_data_tev[hm]);
394                sscanf(str4,"%*e,%le",&gain_ch4_data_tev[hm]);
395         }
396
397
398         int testint;
399         for(testint=0;testint<3;testint++){
```

```
400                 printf("gain_file_freq_data_tev = %e\n",
                        ↪ gain_file_freq_data_tev[testint]);
401         }
402
403         fclose(ch1_gptr_tev);
404         fclose(ch2_gptr_tev);
405         fclose(ch3_gptr_tev);
406         fclose(ch4_gptr_tev);
407
408         printf("read in TEV gain corrections successfully\n");
409    }
410    else{
411       printf("no separate TEV gain corrections used\n");
412    }
413
414 //GAIN-CORRECTED NOISE OF CIRCUIT COMPONENTS
415 //
    ↪ ********************************************************************************
    ↪
416
417         //Calculate the other sources of noise in the system.
418
419         int freq_cntr = 0;
420
421 //NOTE ABOUT THE E1 VOLTAGE NOISE: gain corrected gain resistor 1 (
    ↪ smaller one) voltage noise (gain of the inverting node that r1
    ↪  sees is actually (r2/r1), not (r2/r1 +1). Now, we cannot
    ↪ exactly measure the gain rolloff from this input, but we
    ↪ assume it is the fractionally same rolloff as in the other
    ↪ input. So if the + input has a gain of 90% of (r2/r1+1), we
    ↪ assume the - input has a gain of 90% of (r2/r1). Therefore,
    ↪ gain-corrected e1 will be x, where (gain_correction_measured/(
    ↪ r2/r1+1) = x/(r2/r1)), or x = (r2/r1)*gain_correction_measured
    ↪ /gainz
422         double *ch1_indep_noise_pwr;//gain corrected opamp current
                ↪ noise acting on source impedence magnitude
423         double *ch2_indep_noise_pwr;
424         double *ch3_indep_noise_pwr;
```

```
425         double *ch4_indep_noise_pwr;
426
427         ch1_indep_noise_pwr = (double *)malloc(sizeof(double)*(
                ↪ segment_len/2+1));
428         ch2_indep_noise_pwr = (double *)malloc(sizeof(double)*(
                ↪ segment_len/2+1));
429         ch3_indep_noise_pwr = (double *)malloc(sizeof(double)*(
                ↪ segment_len/2+1));
430         ch4_indep_noise_pwr = (double *)malloc(sizeof(double)*(
                ↪ segment_len/2+1));
431
432         double g1;
433         double g2;
434
435         for(freq_cntr = 0;freq_cntr<(segment_len/2 + 1);freq_cntr++){
                ↪ //number of frequencies dictated by welch segment
                ↪ length
436
437             if(raw_data == 1){
438                     ch1_indep_noise_pwr[freq_cntr] = 0;//no
                            ↪ subtraction of circuit component noise
439                     ch2_indep_noise_pwr[freq_cntr] = 0;//no
                            ↪ subtraction of circuit component noise
440                     ch3_indep_noise_pwr[freq_cntr] = 0;//no
                            ↪ subtraction of circuit component noise
441                     ch4_indep_noise_pwr[freq_cntr] = 0;//no
                            ↪ subtraction of circuit component noise
442             }
443             else if(raw_data == 0){
444                     //freqq = freq_cntr*(samp_rate/2)/(segment_len/2
                            ↪  +1);//The frequency of the future
                            ↪ calculated power spectrum
445
446                     // all sources of noise to be subtracted from
                            ↪ measured PSD. 4*Rs*kb*T*G^2 = (measured
                            ↪ PSD) - (stuff), where (stuff) =
                            ↪ chx_indep_noise_pwr. This is not
                            ↪ extremely computationally efficient, but
```

```
                    ↪ easier to error check.
447
448                 //ch1
449                 ch1_indep_noise_pwr[freq_cntr] =
                    ↪ external_noise_calc(gain_ch1_data_interp[
                    ↪ freq_cntr],daq, r4, temp, in, r3, gainz2,
                    ↪  en, zout, r2, r1, gainz1, r_of_chans[0]
                    ↪ );
450                 //ch2
451                 ch2_indep_noise_pwr[freq_cntr] =
                    ↪ external_noise_calc(gain_ch2_data_interp[
                    ↪ freq_cntr],daq, r4, temp, in, r3, gainz2,
                    ↪  en, zout, r2, r1, gainz1, r_of_chans[1]
                    ↪ );
452                 //ch3
453                 ch3_indep_noise_pwr[freq_cntr] =
                    ↪ external_noise_calc(gain_ch3_data_interp[
                    ↪ freq_cntr],daq, r4, temp, in, r3, gainz2,
                    ↪  en, zout, r2, r1, gainz1, r_of_chans[2]
                    ↪ );
454                 //ch4
455                 ch4_indep_noise_pwr[freq_cntr] =
                    ↪ external_noise_calc(gain_ch4_data_interp[
                    ↪ freq_cntr],daq, r4, temp, in, r3, gainz2,
                    ↪  en, zout, r2, r1, gainz1, r_of_chans[3]
                    ↪ );
456
457
458             }
459             else{
460                 printf("ERROR: raw_data must be either 0 or 1\n"
                    ↪ );
461                 exit(-1);
462             }
463         }
464
465
466
```

```
467 //INITIALIZE NSD AND TEV DATA FILES
468 //
    ↪ ********************************************************************************
    ↪
469         //Initialize the files for NSD and TEV data
470         FILE *fp_on;
471         FILE *fp_off;
472         FILE *fp_tev;
473    FILE *fp_Ron;
474    FILE *fp_Roff;
475         fp_on = fopen(filename_nsd_on,"w+");
476         fp_off = fopen(filename_nsd_off,"w+");
477         fp_tev = fopen(filename_tev,"w+");
478         fp_Ron = fopen(filename_R_on,"w+");
479         fp_Roff = fopen(filename_R_off,"w+");
480         if(fp_on == NULL){
481                printf("trouble creating on data file\n");
482                exit(-1);
483         }
484
485         if(fp_off == NULL){
486                printf("trouble creating off data file\n");
487                exit(-1);
488         }
489
490         if(fp_off == NULL){
491                printf("trouble creating tev data file\n");
492                exit(-1);
493         }
494
495         if(fp_Ron == NULL){
496                printf("trouble creating R on data file\n");
497                exit(-1);
498         }
499
500         if(fp_Roff == NULL){
501                printf("trouble creating R off data file\n");
502                exit(-1);
```

```
503         }
504
505
506         fclose(fp_on);
507         fclose(fp_off);
508         fclose(fp_tev);
509         fclose(fp_Ron);
510         fclose(fp_Roff);
511
512
513         printf("starting\n");
514
515 //INITIALIZING A BUNCH OF STUFF
516 //
    ↪ ********************************************************************************
    ↪
517
518         char ch_data_filename[100];
519         FILE *ch_file_ptr;
520         long filelen;//length of file, long is an int that can be very
                ↪  large
521         int array_len;//length of array
522         double *NSYNC; //data goes into this array
523         int *on_starts;
524         on_starts = (int *)malloc(1*sizeof(int));
525         int *off_starts;
526         off_starts = (int *)malloc(1*sizeof(int));
527         //printf("made the new mallocs\n");
528         double sub_val;
529         int jt;
530         int ons = 0;//index for location of on starts
531         int offs = 0;//index for location of off starts
532         double *data4plot;
533         int vally;
534         fftw_complex * ch_on_data;
535         fftw_complex * ch_off_data;
536         int k;
537         //int z;
```

```
538         int ind_on;
539         int ind_off;
540         int cntr2;
541         int zz;
542         double max_psd;
543         double min_psd;
544 
545         //Spacing of fft frequencies
546         double freq_spacing = (samp_rate/2.0)/(segment_len/2.0+1.0);
547         int freq_min_ind = (int)(freq_min/freq_spacing);
548         int freq_max_ind = (int)(freq_max/freq_spacing);
549 
550         double avg_PSD;
551         double avg_NSD;
552 
553 
554 //INITIALIZING FFTW PLANS AND ARRAYS
555 //
    ↪ ********************************************************************************
    ↪ 
556 
557         fftw_plan plan_segment;
558         //Getting all of the Welch variables ready
559         fftw_complex *segment_data_array;//length of each welch
            ↪ segment length the fft
560         segment_data_array = (fftw_complex *)fftw_malloc((segment_len)
            ↪ *sizeof(fftw_complex));
561 
562         //Getting all of the thermoelectric voltage variables ready
563         int tev_ind=0;
564         double max_tev;
565         int max_tev_ind;
566         fftw_plan plan_segment_tev;
567         fftw_complex *segment_tev_array;
568         fftw_complex *segment_tev_fft_array;
569         double *segment_tev_fft_array_norm;
570         if(is_flat==0 || getRconst == 1){
571                 //Getting all of the thermoelectric voltage variables
```

```
                ↪ ready
572             segment_tev_array = (fftw_complex *)fftw_malloc((
                    ↪ segment_len_4fft)*sizeof(fftw_complex));
573             segment_tev_fft_array = (fftw_complex *)fftw_malloc(
                    ↪ sizeof(fftw_complex)*(segment_len_4fft));
574             printf("Starting TEV fftw plan\n");
575             plan_segment_tev = fftw_plan_dft_1d(segment_len_4fft,
                    ↪ segment_tev_array,segment_tev_fft_array,
                    ↪ FFTW_FORWARD,FFTW_ESTIMATE);//
576 //change to   FFTW_MEASURE eventually
577             printf("Finished TEV fftw plan\n");
578             segment_tev_fft_array_norm = (double *)malloc(sizeof(
                    ↪ double)*segment_len_4fft);
579         }
580
581         //PSD data array
582         double *psd_cumulative_array;
583         psd_cumulative_array = (double *)calloc((segment_len/2+1),
                ↪ sizeof(double));//calloc initializes array to zero
584
585         int segment_counter = 0; //number of segments
586
587         fftw_complex *segment_fft_array;//the array the welch segment
                ↪ will be ffted into
588
589         segment_fft_array = (fftw_complex *)fftw_malloc(sizeof(
                ↪ fftw_complex)*(segment_len));
590
591         printf("planning the segment\n");
592         plan_segment = fftw_plan_dft_1d(segment_len,
                ↪ segment_data_array,segment_fft_array,FFTW_FORWARD,
                ↪ FFTW_ESTIMATE);//change to FFTW_MEASURE eventually
593
594
595         //The fftw initialization for extracting the resistance
                ↪ signals
596         int r_cntr;
597         fftw_plan plan_resistance;
```

```
598         fftw_complex *resistance_array_time;
599         fftw_complex *resistance_array_fftd;
600 
601         resistance_array_time = (fftw_complex *)fftw_malloc((
                ↪ r_segment_len)*sizeof(fftw_complex));
602         resistance_array_fftd = (fftw_complex *)fftw_malloc((
                ↪ r_segment_len)*sizeof(fftw_complex));
603 
604     double *resistance_array_fftd_norm;
605     resistance_array_fftd_norm = (double *)malloc((r_segment_len/2.0 +
            ↪  1.0)*sizeof(double));
606 
607         plan_resistance = fftw_plan_dft_1d(r_segment_len,
                ↪ resistance_array_time, resistance_array_fftd,
                ↪ FFTW_FORWARD,FFTW_ESTIMATE);
608 
609         double *r_array_4avg_on;//holds the R data of each individual
                ↪ segment, and is averaged at the end
610     double *r_array_4avg_off;
611         r_array_4avg_on = (double *)malloc(sizeof(double)*3000);//this
                ↪  will be realloced later
612         r_array_4avg_off = (double *)malloc(sizeof(double)*3000);
613         int numRon = 0;
614         int numRoff = 0;
615 
616         //variables for storing the amplitude of the resistance signal
617         double max_valueR;
618         double max_indexR;
619 
620         double R_avg_on=0;
621     double R_avg_off=0;
622 
623 //BLACKMANHARRIS WINDOW AND NORMALIZATION FACTOR
624 //
    ↪ ********************************************************************************
    ↪ 
625 
626         //The blackmanharris PSD window normalization
```

```
627         //Make the blackmanharris window array
628         double *blackmanharris_array;
629         blackmanharris_array = (double *)malloc(sizeof(double)*(
                ↪ segment_len));
630         for(zz=0;zz<segment_len;zz++){
631                 blackmanharris_array[zz] = blackmanharris(zz,
                        ↪ segment_len);
632         }
633
634         double window_norm = 0.0;
635         printf("making the window norm\n");
636         for(zz=0;zz<segment_len;zz++){
637                 window_norm = window_norm + blackmanharris_array[zz]*
                        ↪ blackmanharris_array[zz];//for fft
638         }
639
640         double window_norm_correct = 0.0;
641         printf("making the window norm\n");
642         for(zz=0;zz<segment_len;zz++){
643                 window_norm_correct = window_norm_correct +
                        ↪ blackmanharris_array[zz]*blackmanharris_array[zz
                        ↪ ];
644         }
645
646         window_norm_correct = window_norm_correct*samp_rate;
647
648
649         //int zz;
650         int on_cntr = 0;
651         int off_cntr = 0;
652         long cntr=0;
653         int cum_data=0;
654         int segs_left=1;
655         int is_on_ch = 0;
656         int scan_num;
657
658
659
```

```
660 //BEGIN DATA READ IN
661 //
    ↪ ********************************************************************************
    ↪
662
663
664 //looping over the number of scans at each power or wavelength (a for
    ↪  loop inside this one loops over the channels)
665         for(scan_num = 0;scan_num<num_scans;scan_num++){
666
667 //If a flat scan, slightly different route:
668                 if(is_flat == 1){
669                         sprintf(ch_data_filename,"%sch1_scan%d.bin",
                            ↪ data_path,scan_num+1);
670                         printf("Data is not chopped\n");
671                 }
672                 else if(is_flat == 0){
673 //Open the chopper data
674                         sprintf(ch_data_filename,"%sch5_scan%d.bin",
                            ↪ data_path,scan_num+1);
675                         printf("Data is chopped\n");
676                 }
677                 else{
678                         printf("Incorrect value for is_flat\n");
679                         exit(0);
680                 }
681
682                 //Open file
683                 ch_file_ptr = fopen(ch_data_filename,"rb");
684                 if(ch_file_ptr == NULL){
685                         printf("Problem opening first data file\n");
686                         exit(-1);
687                 }
688                 printf("opened the first data file\n");
689                 //Read the binary file data contents into an array
                    ↪ ready for fft
690                 //initialize array for first scan. All arrays must be
                    ↪ the same size.
```

```
691
692             if(scan_num == 0){//just do this stuff the first time,
                    ↪ since all arrays are the same length
693                 fseek(ch_file_ptr, 0, SEEK_END); //Go to end of
                        ↪ file
694                 printf("found end of file\n");
695                 filelen = ftell(ch_file_ptr); //say what
                        ↪ position is the end
696                 rewind(ch_file_ptr); //Put pointer back to
                        ↪ beginning of file
697                 printf("filelen %ld\n",filelen);
698                 printf("about to malloc\n");
699                 array_len = filelen/8; //assumes data is doubles
                        ↪ , so 8 bytes each
700                 NSYNC = (double *)malloc((filelen+1)*sizeof(
                        ↪ double));//may need to do filelen+1, don'
                        ↪ t know if binary doubles have a null
                        ↪ character, don't think so
701                 printf("mallocked\n");
702
703                 //Again, the different route for a flat scan
704                 if(is_flat ==1){
705                        ch_off_data = (fftw_complex*)malloc(
                            ↪ sizeof(fftw_complex)*array_len);
706                        //ch_on_data will be unused
707                 }
708                 else if(is_flat == 0){
709                 //Determine amount of space for the "on" and "
                        ↪ off" data vectors. -> Overallocating so
                        ↪ we don't run out of space (we know it
                        ↪ must be length of total data/2 +1 period
                        ↪ maximum, so we will do that, and can cut
                        ↪ later
710                     ch_on_data = (fftw_complex*)fftw_malloc(
                            ↪ sizeof(fftw_complex)*((array_len/2)
                            ↪ +300000)); //200000 is number of
                            ↪ points in 1 period of 10Hz with 2e6
                            ↪  S/s
```

```
711                             ch_off_data = (fftw_complex*)fftw_malloc(
                                    ↪ sizeof(fftw_complex)*((array_len/2)
                                    ↪ +300000));
712                     }
713                     else{
714                             printf("error with is_flat value\n");
715                             exit(0);
716                     }
717             }
718
719             fread(NSYNC,filelen,1,ch_file_ptr);
720             fclose(ch_file_ptr);
721             printf("read in data successfully\n");
722
723
724 //CONVERT CHOPPER DATA TO +1 AND -1
725 //
    ↪ ********************************************************************************
    ↪
726             if(is_flat == 0){
727                     //find mean of data (or first 1/3 of data) so
                            ↪ the "on" will be >0 and the "off" will be
                            ↪  <0
728                     double mean_val = 0.0;
729                     int tj;
730                     for(tj=0;tj<array_len;tj++){
731                             mean_val = mean_val + NSYNC[tj];
732                     }
733                     mean_val = mean_val/((array_len));
734                     printf("mean_val is: %f\n",mean_val);
735
736                     //find the on/off regions from the chopper
                            ↪ signal. Make everything +/-1, then take
                            ↪ the difference. Note: position 2-position
                            ↪  1 = position 0. So essentially I am
                            ↪ cutting off the first 2 parts of the
                            ↪ array.
737
```

```
738                     ons = 0;
739                     offs = 0;
740                     for(jt = 1;jt<array_len;jt++){//yes, starting
                            ↪ from 1, not 0
741                         //1 ahead
742                         if(jt == 1){//changed 20190407
743                             if((NSYNC[jt-1]-mean_val)<0.0 && (
                                    ↪ NSYNC[jt]-mean_val)<0.0){//
                                    ↪ changed 20190407 to sample the
                                    ↪ point and the next point, works
                                    ↪  for noisy data
744                                 NSYNC[jt-1]=-1.0;
745                             }
746                             else if((NSYNC[jt-1]-mean_val)>0.0 &&
                                    ↪ (NSYNC[jt]-mean_val)>0.0){
747                                 NSYNC[jt-1]=1.0;
748                             }
749                             else{
750                                 NSYNC[jt-1]=0.0;
751                             }
752                         }
753
754                         //2 ahead
755                         //4/7/2019 - smooths slightly noisy
                                ↪ chopper data (assuming problem
                                ↪ spike only lasts one data point)
756
757                         //case 1: random spike above zero from "
                                ↪ light off" (low HIGH low)
758                         if((NSYNC[jt]-mean_val)>0.0 && (NSYNC[jt
                                ↪ -1]-mean_val)<0.0 && (NSYNC[jt+1]-
                                ↪ mean_val)<0.0){
759                             NSYNC[jt]=-1.0;//Is a random high
                                    ↪ spike when the light is off
760                             //printf("case 1\n");
761                         }
762                         //case 2: random spike below zero from "
                                ↪ light on" (high LOW high)
```

```
763                     else if((NSYNC[jt]-mean_val)<0.0 && (
                            ↪ NSYNC[jt-1]-mean_val)>0.0 && (NSYNC
                            ↪ [jt+1]-mean_val)>0.0){
764                         NSYNC[jt]=1.0;//Is a random low
                                ↪ spike when the light is off
765                         //printf("case 2\n");
766                     }
767                     //case 3: data point right before "on" (
                            ↪ low LOW high)
768                     else if((NSYNC[jt]-mean_val)<0.0 && (
                            ↪ NSYNC[jt-1]-mean_val)<0.0 && (NSYNC
                            ↪ [jt+1]-mean_val)>0.0){
769                         NSYNC[jt]=-1.0;//right before step
                                ↪  up
770                     }
771                     //case 4: data point right after "on" (
                            ↪ low HIGH high)
772                     else if((NSYNC[jt]-mean_val)>0.0 && (
                            ↪ NSYNC[jt-1]-mean_val)<0.0 && (NSYNC
                            ↪ [jt+1]-mean_val)>0.0){
773                         NSYNC[jt]=1.0;//right after step
                                ↪ up
774                     }
775                     //case 5: data point right before "off" (
                            ↪ high HIGH low)
776                     else if((NSYNC[jt]-mean_val)>0.0 && (
                            ↪ NSYNC[jt-1]-mean_val)>0.0 && (NSYNC
                            ↪ [jt+1]-mean_val)<0.0){
777                         NSYNC[jt]=1.0;//right before step
                                ↪ down
778                     }
779                     //case 6: data point right after "off" (
                            ↪ high LOW low)
780                     else if((NSYNC[jt]-mean_val)<0.0 && (
                            ↪ NSYNC[jt-1]-mean_val)>0.0 && (NSYNC
                            ↪ [jt+1]-mean_val)<0.0){
781                         NSYNC[jt]=-1.0;//right after step
                                ↪ up
```

```
782                         }
783                         //case 7: data point in the low region (
                                ↪ low LOW low)
784                         else if((NSYNC[jt]-mean_val)<0.0 && (
                                ↪ NSYNC[jt-1]-mean_val)<0.0 && (NSYNC
                                ↪ [jt+1]-mean_val)<0.0){
785                             NSYNC[jt]=-1.0;//low
786                         }
787                         //case 8: data point in the low region (
                                ↪ low LOW low)
788                         else if((NSYNC[jt]-mean_val)>0.0 && (
                                ↪ NSYNC[jt-1]-mean_val)>0.0 && (NSYNC
                                ↪ [jt+1]-mean_val)>0.0){
789                             NSYNC[jt]=1.0;//low
790                         }
791                         else{
792                             NSYNC[jt]=NSYNC[jt-1];//if none of
                                    ↪  the above, like it equals
                                    ↪ zero, just make it the same
                                    ↪ as the point before
793                         }
794                         //Where you find the indices of break
                                ↪ points
795
796                         sub_val = NSYNC[jt]-NSYNC[jt-1];
797
798                         if(sub_val>0.0){//change 4/7/2019 from
                                ↪ 0.0 to 1.1
799                             on_starts = (int *)realloc(
                                    ↪ on_starts,(ons+1)*sizeof(int
                                    ↪ ));
800                             on_starts[ons] = jt;//changed
                                    ↪ 4/7/2019 from jt-1 to jt,
                                    ↪ should make no difference
801                             ons++;
802                         }
803                         else if(sub_val<0.0){
804                             off_starts = (int *)realloc(
```

```
                                    ↪ off_starts,(offs+1)*sizeof(
                                    ↪ int));
805                                 if(off_starts == NULL){
806                                         printf("memory allocation
                                            ↪ failed\n");
807                                         exit(1);
808                                 }
809                                 off_starts[offs] = jt;//changed
                                    ↪ 4/7/2019 from jt-1 to jt
810                                 offs++;
811                         }
812                         else{
813                                 //printf("is zero\n");
814                         }
815
816                 }
817         }
818
819         printf("ons = %d, offs = %d\n",ons,offs);
820
821
822 //PLOTTING PERFECT CHOPPER PLUS DATA CHANNEL 3
823 //
    ↪ ********************************************************************************
    ↪
824         //Plotting the transition location
825         if(plot_data == 1){
826
827                 sprintf(ch_data_filename,"%sch3_scan1.bin",
                        ↪ data_path);
828                 ch_file_ptr = fopen(ch_data_filename,"rb");
829
830
831                 data4plot = (double *)malloc(ind2plot*sizeof(
                        ↪ double));
832                 fread(data4plot,ind2plot*8,1,ch_file_ptr); //8
                        ↪ assumes the data is a double, so 8 bytes
                        ↪ each
```

```
833                     fclose(ch_file_ptr);
834
835
836                     for(vally=0;vally<10;vally++)
837                             printf("data is ch3, ch4: %f, %f\n",
                                    ↪ data4plot[vally],NSYNC[vally]);
838
839                     plotmultiplot(ind2plot,NSYNC,data4plot);
840                     exit(1);
841             }
842
843
844 //LOOP OVER EACH CHANNEL (k)
845 //
    ↪ ********************************************************************************
    ↪
846             //Looping over the channels
847             for(k=0; k<num_chans; k++){
848
849                     is_on_ch = 0;
850                     on_cntr = 0;
851                     off_cntr = 0;
852                     cntr = 0;
853                     cum_data = 0;
854                     segs_left = 1;
855
856                     //read in data - containing file into NSYNC
                            ↪ array
857                     sprintf(ch_data_filename,"%sch%d_scan%d.bin",
                            ↪ data_path,k+1,scan_num+1);
858                     ch_file_ptr = fopen(ch_data_filename,"rb");
859                     if(ch_file_ptr == NULL){
860                             printf("error reading files channel loop\
                                    ↪ n");
861                             exit(0);
862                     }
863
864                     fread(NSYNC,filelen,1,ch_file_ptr);
```

```
865                     fclose(ch_file_ptr);
866
867 //THERMOELECTRIC VOLTAGE ANALYSIS
868 //
    ↪ ********************************************************************************
    ↪
869                     //Here we find the thermoelectric voltage, if
                            ↪ not a control spectrum
870                     if(is_flat == 0 || getRconst == 1){
871
872                             //fft NSYNC - make a plan beforehand and
                                    ↪ a place to put the fft
873                             for(tev_ind = 0;tev_ind<segment_len_4fft;
                                    ↪ tev_ind++){
874                                     segment_tev_array[tev_ind] = NSYNC
                                            ↪ [tev_ind];
875                             }
876
877                             fftw_execute(plan_segment_tev);//fft data
                                    ↪  in segment_tev_fft_array
878
879                             //properly normalize (is no window, so
                                    ↪ just divide by segment_len_4fft, x
                                    ↪ by 2 because 2-sided).
880                             for(tev_ind = 0;tev_ind<segment_len_4fft;
                                    ↪ tev_ind++){
881                                     segment_tev_fft_array_norm[tev_ind
                                            ↪ ] = 2.0*sqrt(creal(
                                            ↪ segment_tev_fft_array[
                                            ↪ tev_ind])*creal(
                                            ↪ segment_tev_fft_array[
                                            ↪ tev_ind]) + cimag(
                                            ↪ segment_tev_fft_array[
                                            ↪ tev_ind])*cimag(
                                            ↪ segment_tev_fft_array[
                                            ↪ tev_ind]))/((double)(
                                            ↪ segment_len_4fft));
882                             }
```

```
883
884                         //Plot the fft if desired
885                         if(plot_fft == 1){
886                                 plotlogplotfft(segment_len_4fft
                                        ↪ /2+1,samp_rate/2,
                                        ↪ segment_tev_fft_array_norm);
887                                 printf("press ENTER to continue\n"
                                        ↪ );
888                                 getchar();
889                         }
890
891                         //ind_chop_rate = chop_rate/((samp_rate
                            ↪ /2.0)/(segment_len_4fft/2.0+1.0));
892                         max_tev = 0.0;//initialize to zero to
                            ↪ find max voltage peak
893                         max_tev_ind = 0;//find index of peak
894                         for(tev_ind = (int)((chop_rate-10)/((
                            ↪ samp_rate/2.0)/(segment_len_4fft
                            ↪ /2.0+1.0)));tev_ind<(int)((
                            ↪ chop_rate+10)/((samp_rate/2.0)/(
                            ↪ segment_len_4fft/2.0+1.0)));tev_ind
                            ↪ ++){//search for the peak +/- 5 Hz
                            ↪ around the chopper rate (in case
                            ↪ the frequency was actually 43.1 Hz
                            ↪ instead of 43 Hz, just to get
                            ↪ better accuracy)
895                                 if(segment_tev_fft_array_norm[
                                        ↪ tev_ind]>max_tev){
896                                         max_tev =
                                                ↪ segment_tev_fft_array_norm
                                                ↪ [tev_ind];//find max
                                                ↪ voltage in this range
897                                         max_tev_ind = tev_ind;
898                                 }
899                         }
900                         printf("Peak voltage is %e V at %f Hz\n",
                            ↪ max_tev,max_tev_ind*((samp_rate
                            ↪ /2.0)/(segment_len_4fft/2.0+1.0)));
```

```
901
902                         //multiply the peak by 2*pi/4 (for TEV
                               ↪ extraction, not R extraction) - the
                               ↪  2 is because we want the peak to
                               ↪ peak
903             if(is_flat==0){//only for TEV extraction, not R extraction
904                         printf("max_tev before alteration is %e\
                                ↪ n",max_tev);
905                         max_tev = max_tev * 2.0 * PI / 4.0;
906             }
907
908             //analyze differently if using separate gain for TEV
909             if(tev_gain_meas == 0){
910
911                         //multiply by the gain of the amplitude
                               ↪ at the frequency of interest
912                           if(k==0){//if chan 1
913                                 max_tev = max_tev / linear_interpy
                                       ↪ (max_tev_ind*((samp_rate
                                       ↪ /2.0)/(segment_len_4fft
                                       ↪ /2.0+1.0)),
                                       ↪ gain_file_freq_data,
                                       ↪ gain_ch1_data, samp_rate
                                       ↪ /2.0);
914                         }
915                         else if(k == 1){//if chan 2
916                                 max_tev = max_tev / linear_interpy
                                       ↪ (max_tev_ind*((samp_rate
                                       ↪ /2.0)/(segment_len_4fft
                                       ↪ /2.0+1.0)),
                                       ↪ gain_file_freq_data,
                                       ↪ gain_ch2_data, samp_rate
                                       ↪ /2.0);
917                         }
918                         else if(k == 2){//if chan 3
919                                 max_tev = max_tev / linear_interpy
                                       ↪ (max_tev_ind*((samp_rate
                                       ↪ /2.0)/(segment_len_4fft
```

```
                                        ↪ /2.0+1.0)),
                                        ↪ gain_file_freq_data,
                                        ↪ gain_ch3_data, samp_rate
                                        ↪ /2.0);
920                             }
921                             else if(k == 3){//if chan 4
922                                     max_tev = max_tev / linear_interpy
                                        ↪ (max_tev_ind*((samp_rate
                                        ↪ /2.0)/(segment_len_4fft
                                        ↪ /2.0+1.0)),
                                        ↪ gain_file_freq_data,
                                        ↪ gain_ch4_data, samp_rate
                                        ↪ /2.0);
923                             }
924                             else{
925                                     printf("TEV extraction - too many
                                        ↪ channels, you can only have
                                        ↪  4 signal channels +
                                        ↪ chopper signal at this time
                                        ↪ .\n");
926                                     exit(0);
927                             }
928             }
929             else if(tev_gain_meas == 1){
930
931                             //multiply by the gain of the amplitude
                                   ↪ at the frequency of interest
932                              if(k==0){//if chan 1
933                                     max_tev = max_tev / linear_interpy
                                        ↪ (max_tev_ind*((samp_rate
                                        ↪ /2.0)/(segment_len_4fft
                                        ↪ /2.0+1.0)),
                                        ↪ gain_file_freq_data_tev,
                                        ↪ gain_ch1_data_tev,
                                        ↪ samp_rate/2.0);
934                             }
935                             else if(k == 1){//if chan 2
936                                     max_tev = max_tev / linear_interpy
```

```
                                        ↪ (max_tev_ind*((samp_rate
                                        ↪ /2.0)/(segment_len_4fft
                                        ↪ /2.0+1.0)),
                                        ↪ gain_file_freq_data_tev,
                                        ↪ gain_ch2_data_tev,
                                        ↪ samp_rate/2.0);
937                                 }
938                                 else if(k == 2){//if chan 3
939                                         max_tev = max_tev / linear_interpy
                                            ↪ (max_tev_ind*((samp_rate
                                            ↪ /2.0)/(segment_len_4fft
                                            ↪ /2.0+1.0)),
                                            ↪ gain_file_freq_data_tev,
                                            ↪ gain_ch3_data_tev,
                                            ↪ samp_rate/2.0);
940                                 }
941                                 else if(k == 3){//if chan 4
942                                         max_tev = max_tev / linear_interpy
                                            ↪ (max_tev_ind*((samp_rate
                                            ↪ /2.0)/(segment_len_4fft
                                            ↪ /2.0+1.0)),
                                            ↪ gain_file_freq_data_tev,
                                            ↪ gain_ch4_data_tev,
                                            ↪ samp_rate/2.0);
943                                 }
944                                 else{
945                                         printf("TEV extraction - too many
                                            ↪ channels, you can only have
                                            ↪  4 signal channels +
                                            ↪ chopper signal at this time
                                            ↪ .\n");
946                                         exit(0);
947                                 }
948                         }
949                         else{
950                            printf("tev_gain_meas must be 0 or 1, failed during TEV
                                ↪ analysis step\n");
951                            exit(0);
```

```
952             }
953                             //save this data in a file which also
                                    ↪ must be made before and terminated.
954                             fp_tev = fopen(filename_tev,"a");
955                             fprintf(fp_tev,"%le\t",max_tev);
956                             fclose(fp_tev);
957
958                         }
959
960
961 //SEPARATE ON AND OFF DATA
962 //
    ↪ ********************************************************************************
    ↪
963                         //Here I separate the on and off data into the
                                ↪ different arrays to fft, if not a control
                                ↪  spectrum
964                         if(is_flat == 0){
965
966                             for(ind_on = 0;ind_on < ons;ind_on++){//
                                    ↪ Loop over all the on transitions
967                                 //Basically determine which index
                                        ↪ follows
968                                 for(cntr2 = 0;cntr2<(fabs(
                                        ↪ off_starts[5]-on_starts[5])-
                                        ↪ right_cut-left_cut);cntr2++)
                                        ↪ {//Will cut correctly +/- a
                                        ↪ few spots on each side
969                                     ch_on_data[on_cntr] = NSYNC[
                                            ↪ on_starts[ind_on]+
                                            ↪ right_cut+cntr2];
970                                     on_cntr++;
971                                     //R array, only for channels
                                            ↪  2 and 3
972                                     if(k==1 || k==2){
973                                         if(cntr2<
                                                ↪ r_segment_len)
                                                ↪ {//We don't
```

```
                                        ↪ want to
                                        ↪ overflow the
                                        ↪ array. If it
                                        ↪ underflows,
                                        ↪ there will be
                                        ↪ zeros at the
                                        ↪ end. Will look
                                        ↪  into this
                                        ↪ futher.
974                                     resistance_array_time
                                            ↪ [cntr2]
                                            ↪  =
                                            ↪ NSYNC[
                                            ↪ on_starts
                                            ↪ [ind_on
                                            ↪ ]+
                                            ↪ right_cut
                                            ↪ +cntr2
                                            ↪ ];
975                                 }
976                             }
977                         }
978                         //printf("cntr2 is %d\n",cntr2);
979                         //printf("r_segment_len is %f\n",
                                ↪ r_segment_len);
980
981                         //Only for channels 2 and 3 do we
                                ↪ look for the resistance peak
982                         if(k==1 || k==2){
983                             //Here we fft the resistance
                                    ↪  array and find the
                                    ↪ peak at the correct
                                    ↪ resistance
984                             fftw_execute(plan_resistance
                                    ↪ );
985
986                             //Properly normalize the fft
                                    ↪  (2*absolute value/
```

```
                                    ↪ sample num). .
987                                 for(r_cntr=0;r_cntr<(
                                        ↪ r_segment_len/2.0)
                                        ↪ +1.0;r_cntr++){
988                                         resistance_array_fftd_norm
                                                ↪ [r_cntr] =
                                                ↪ 2.0*sqrt(creal
                                                ↪ (
                                                ↪ resistance_array_fftd
                                                ↪ [r_cntr])*
                                                ↪ creal(
                                                ↪ resistance_array_fftd
                                                ↪ [r_cntr]) +
                                                ↪ cimag(
                                                ↪ resistance_array_fftd
                                                ↪ [r_cntr])*
                                                ↪ cimag(
                                                ↪ resistance_array_fftd
                                                ↪ [r_cntr]))/((
                                                ↪ double)(
                                                ↪ r_segment_len)
                                                ↪ );
989                                 }
990
991                                 //Plot the fft if desired
992                                 if(plot_fft_R == 1){
993                                         plotlogplotfft(
                                                ↪ r_segment_len
                                                ↪ /2+1,samp_rate
                                                ↪ /2,
                                                ↪ resistance_array_fftd_norm
                                                ↪ );
994                                         printf("R fft plot,
                                                ↪ press ENTER to
                                                ↪  continue\n");
995                                         getchar();
996                                 }
997
```

```
998                                 //Now we find the peak of
                                        ↪ the resistance signal
                                        ↪ . (for ch2 or ch3)
999                                 if(k==1){
1000                                    FindAmpAtFreq(ch2_r_freq
                                            ↪ , r_segment_len
                                            ↪ /2.0+1.0,
                                            ↪ samp_rate/2.0,
                                            ↪ resistance_array_fftd_norm
                                            ↪ , &max_valueR, &
                                            ↪ max_indexR);
1001                                }
1002                                else if(k==2){
1003                                    FindAmpAtFreq(ch3_r_freq
                                            ↪ , r_segment_len
                                            ↪ /2.0+1.0,
                                            ↪ samp_rate/2.0,
                                            ↪ resistance_array_fftd_norm
                                            ↪ , &max_valueR, &
                                            ↪ max_indexR);
1004                                }
1005                                //printf("Peak R voltage (on
                                        ↪ ) is %e V at %f Hz\n
                                        ↪ ",max_valueR,
                                        ↪ max_indexR*((
                                        ↪ samp_rate/2.0)/(
                                        ↪ r_segment_len
                                        ↪ /2.0+1.0)));
1006
1007                                //Save this data in an array
                                        ↪  which will be
                                        ↪ averaged later
1008                                if(numRon>3000){//reallocate
                                        ↪  length if needed
1009                                    r_array_4avg_on = (
                                            ↪ double *)realloc(
                                            ↪ r_array_4avg_on,(
                                            ↪ numRoff+1)*sizeof
```

```
                                        ↪ (double));
1010                                }
1011                                r_array_4avg_on[ind_on] =
                                        ↪ max_valueR;
1012                                numRon++;
1013                        }
1014                }
1015
1016                for(ind_off = 0;ind_off < offs;ind_off++)
                        ↪ {
1017                        for(cntr2 = 0;cntr2<(fabs(
                                ↪ off_starts[5]-on_starts[5])-
                                ↪ right_cut-left_cut);cntr2++)
                                ↪ {
1018                                ch_off_data[off_cntr] =
                                        ↪ NSYNC[off_starts[
                                        ↪ ind_off]+right_cut+
                                        ↪ cntr2];
1019                                off_cntr++;
1020
1021                              if(k==1 || k==2){
1022                                        if(cntr2<
                                                ↪ r_segment_len)
                                                ↪ {//We don't
                                                ↪ want to
                                                ↪ overflow the
                                                ↪ array. If it
                                                ↪ underflows,
                                                ↪ there will be
                                                ↪ zeros at the
                                                ↪ end. Will look
                                                ↪  into this
                                                ↪ futher.
1023                                            resistance_array_time
                                                    ↪ [cntr2]
                                                    ↪  =
                                                    ↪ NSYNC[
                                                    ↪ off_starts
```

```
                                                                    ↪ [
                                                                    ↪ ind_off
                                                                    ↪ ]+
                                                                    ↪ right_cut
                                                                    ↪ +cntr2
                                                                    ↪ ];
1024                                                        }
1025                                                }
1026                                        }
1027                                        //printf("cntr2 is %d\n",cntr2);
1028                                        //printf("r_segment_len is %f\n",
                                                ↪ r_segment_len);
1029
1030
1031    //LOOK FOR RESISTANCE PEAK
1032    //
            ↪ ********************************************************************************
            ↪
1033                                        //Only for channels 2 and 3 do we
                                                ↪ look for the resistance peak
1034                                        if(k==1 || k==2){
1035                                                //Here we fft the resistance
                                                        ↪  array and find the
                                                        ↪ peak at the correct
                                                        ↪ resistance
1036                                                fftw_execute(plan_resistance
                                                        ↪ );
1037
1038                                                //Properly normalize the fft
                                                        ↪  (2*absolute value/
                                                        ↪ sample num). We are
                                                        ↪ putting it back into
                                                        ↪ the fftd array, just
                                                        ↪ the first half. So
                                                        ↪ only the first half
                                                        ↪ of the array should
                                                        ↪ be used from this
                                                        ↪ point on.
```

```
1039                                for(r_cntr=0;r_cntr<(
                                        ↪ r_segment_len/2.0)
                                        ↪ +1.0;r_cntr++){
1040                                    resistance_array_fftd_norm
                                            ↪ [r_cntr] =
                                            ↪ 2.0*sqrt(creal
                                            ↪ (
                                            ↪ resistance_array_fftd
                                            ↪ [r_cntr])*
                                            ↪ creal(
                                            ↪ resistance_array_fftd
                                            ↪ [r_cntr]) +
                                            ↪ cimag(
                                            ↪ resistance_array_fftd
                                            ↪ [r_cntr])*
                                            ↪ cimag(
                                            ↪ resistance_array_fftd
                                            ↪ [r_cntr]))/((
                                            ↪ double)(
                                            ↪ r_segment_len)
                                            ↪ );
1041                                }
1042
1043                                //Plot the fft if desired
1044                                if(plot_fft_R == 1){
1045                                       plotlogplotfft(
                                            ↪ r_segment_len
                                            ↪ /2+1,samp_rate
                                            ↪ /2,
                                            ↪ resistance_array_fftd_norm
                                            ↪ );
1046                                       printf("R fft plot,
                                            ↪ press ENTER to
                                            ↪  continue\n");
1047                                       getchar();
1048                                }
1049
1050                                //Now we find the peak of
```

```
                                    ↪ the resistance signal
                                    ↪ . (for ch2 or ch3)
1051                                if(k==1){
1052                                    FindAmpAtFreq(ch2_r_freq
                                        ↪ , r_segment_len
                                        ↪ /2.0+1.0,
                                        ↪ samp_rate/2.0,
                                        ↪ resistance_array_fftd_norm
                                        ↪ , &max_valueR, &
                                        ↪ max_indexR);
1053                                }
1054                                else if(k==2){
1055                                    FindAmpAtFreq(ch3_r_freq
                                        ↪ , r_segment_len
                                        ↪ /2.0+1.0,
                                        ↪ samp_rate/2.0,
                                        ↪ resistance_array_fftd_norm
                                        ↪ , &max_valueR, &
                                        ↪ max_indexR);
1056                                }
1057                                //printf("Peak R voltage (
                                    ↪ off) is %e V at %f Hz
                                    ↪ \n",max_valueR,
                                    ↪ max_indexR*((
                                    ↪ samp_rate/2.0)/(
                                    ↪ r_segment_len
                                    ↪ /2.0+1.0)));
1058
1059                                //Save this data in an array
                                    ↪  which will be
                                    ↪ averaged later
1060                                if(numRoff>3000){//
                                    ↪ reallocate length if
                                    ↪ needed
1061                                    r_array_4avg_off = (
                                        ↪ double *)realloc(
                                        ↪ r_array_4avg_off
                                        ↪ ,(numRoff+1)*
```

```
                                        ↪ sizeof(double));
1062                                }
1063                                r_array_4avg_off[ind_off] =
                                        ↪ max_valueR;
1064                                numRoff++;
1065                        }
1066                }
1067
1068                if(k==1 || k==2){
1069                        //Here we average the Ron/off data
                            ↪  and save in file.
1070                        //The on data
1071                        for(r_cntr=0;r_cntr<numRon;r_cntr
                            ↪ ++){
1072                            R_avg_on = R_avg_on +
                                ↪ r_array_4avg_on[
                                ↪ r_cntr]/((double)
                                ↪ numRon);
1073                        }
1074                        //The off data
1075                        for(r_cntr=0;r_cntr<numRoff;r_cntr
                            ↪ ++){
1076                            R_avg_off = R_avg_off +
                                ↪ r_array_4avg_off[
                                ↪ r_cntr]/((double)
                                ↪ numRoff);
1077                        }
1078
1079                        //Save the data
1080                        fp_Ron = fopen(filename_R_on,"a");
1081                        fprintf(fp_Ron,"%le\t",R_avg_on);
1082                        fclose(fp_Ron);
1083
1084                        fp_Roff = fopen(filename_R_off,"a"
                            ↪ );
1085                        fprintf(fp_Roff,"%le\t",R_avg_off)
                            ↪ ;
1086                        fclose(fp_Roff);
```

```
1087
1088                                    //Reset values
1089                                    numRoff = 0;
1090                                    numRon = 0;
1091                    R_avg_on = 0;
1092                    R_avg_off = 0;
1093                                }
1094
1095                                printf("split the data into on and off
                                        ↪ portions\n");
1096
1097                                if(plot_errchk==1){
1098                                        printf("plotting first 1 values of
                                                ↪  on and off data\n");
1099                                        int b;
1100                                        for(b=0;b<10;b++){
1101                                                printf("on data: %f, off
                                                        ↪ data: %f\n",creal(
                                                        ↪ ch_on_data[b]),creal(
                                                        ↪ ch_off_data[b]));
1102                                        }
1103                                        printf("plotting on and off data
                                                ↪ for errorchecking\n");
1104                                        printf("plotting on data\n");
1105                                        plotplot(ind2plot,ch_on_data);
1106                                        printf("press ENTER to continue\n"
                                                ↪ );
1107                                        getchar();
1108                                        printf("plotting off data\n");
1109                                        plotplot(ind2plot,ch_off_data);
1110                                        printf("press ENTER to continue\n"
                                                ↪ );
1111                                        getchar();
1112                                }
1113                                printf("on_cntr = %d, off_cntr = %d\n",
                                        ↪ on_cntr,off_cntr);
1114                                if(on_cntr<segment_len){
1115                                        printf("segment_len too large,
```

```
                                        ↪ larger than the on data.\n")
                                        ↪ ;
1116                                    exit(0);
1117                            }
1118                            if(off_cntr<segment_len){
1119                                    printf("segment_len too large,
                                        ↪ larger than the off data.\n"
                                        ↪ );
1120                                    exit(0);
1121                            }
1122
1123                    }
1124
1125
1126                    else{//for flat sprectrum, all data goes into
                        ↪ ch_off_data
1127                        for(ind_off = 0; ind_off < array_len;
                            ↪ ind_off++){
1128                            ch_off_data[ind_off] = NSYNC[
                                ↪ ind_off];
1129                        }
1130                    }
1131
1132
1133 //LOOP OVER THE ON AND OFF DATA
1134 //
    ↪ ********************************************************************************
    ↪
1135
1136                    for(is_on_ch = 0;is_on_ch < 2;){//We first do
                        ↪ the ffts of the on data, then the off
                        ↪ data. 0 is off, 1 is on.
1137
1138
1139 //EXTRACT THE RESISTANCE SIGNAL
1140 //
    ↪ ********************************************************************************
    ↪
```

```
1141                    //if(r_segment_len < off_cntr){
1142                        //printf("r_segment_len is longer
                                ↪ than the number in off array
                                ↪ , for R measurements\n");
1143                        //exit(-1);
1144                    //}
1145                    //for(r_cntr = 0; r_cntr< r_segment_len;
                            ↪ r_cntr++){
1146                        ////read the on or off data into
                                ↪ the array for ffting
1147                        //if(is_on_ch == 0){//read off
                                ↪ data
1148                            //resistance_array_time[
                                    ↪ r_cntr] = ch_off_data
                                    ↪ [r_cntr];
1149                        //}
1150                        //else if(is_on_ch == 1){//read on
                                ↪  data
1151                            //resistance_array_time[
                                    ↪ r_cntr] = ch_on_data[
                                    ↪ r_cntr];
1152                        //}
1153                        //else{
1154                            //printf("problem with
                                    ↪ number of is_on_ch,
                                    ↪ should be only 0 or
                                    ↪ 1\n");
1155                            //exit(1);
1156                        //}
1157                    //}
1158                    ////perform the fft, puts the fft'd
                            ↪ resistance_array_time into
                            ↪ resistance_array_fftd
1159                    //fftw_execute(plan_resistance);
1160
1161                    ////Problem: we can't split up a signal,
                            ↪ patch it together, and get accurate
                            ↪  signal data for a single signal.
```

```
1162
1163
1164                        //}
1165
1166 //SPLIT DATA INTO SEGMENTS AND APPLY WINDOW FUNCTION
1167                                 printf("is_on_ch is: %d (0 is off, 1 is
                                         ↪ on)\n",is_on_ch);
1168                                 segment_counter = 0;
1169                                 segs_left = 1;
1170                                 cum_data = 0;//reinitialize the
                                         ↪ cumulative data counter
1171                                 while(segs_left == 1){
1172
1173                                 //Putting the data into the segment
                                         ↪ allocation, multiplied by
                                         ↪ blackmanharris
1174                                     for(cntr=0;cntr<segment_len;cntr
                                             ↪ ++){
1175                                         //printf("ch_data_array %f\n
                                                 ↪ ", ch_data_array[
                                                 ↪ cum_data]);
1176                                         if(is_on_ch == 0){//read off
                                                 ↪  data
1177                                             segment_data_array[
                                                     ↪ cntr] =
                                                     ↪ ch_off_data[
                                                     ↪ cum_data]*
                                                     ↪ blackmanharris_array
                                                     ↪ [cntr];
1178                                         }
1179                                         else if(is_on_ch == 1){//
                                                 ↪ read on data
1180                                             segment_data_array[
                                                     ↪ cntr] =
                                                     ↪ ch_on_data[
                                                     ↪ cum_data]*
                                                     ↪ blackmanharris_array
                                                     ↪ [cntr];
```

```
1181                                        }
1182                                        else{
1183                                                printf("problem with
                                                    ↪ number of
                                                    ↪ is_on_ch,
                                                    ↪ should be only
                                                    ↪  0 or 1\n");
1184                                                exit(1);
1185                                        }
1186
1187                                        cum_data = cum_data+1;
1188
1189                                }
1190
1191
1192 //DO THE FFT FOR NOISE CALCULATIONS
1193 //
     ↪ ********************************************************************************
     ↪
1194
1195
1196                                //printf("executing segment fft %d
                                        ↪ \n",cum_data);
1197                                fftw_execute(plan_segment);
1198
1199
1200 //DIRECTLY PLOT THE NSD AND GET AVERAGE IF ERROR CHECKING
1201 //
     ↪ ********************************************************************************
     ↪
1202
1203                                //error checking fft
1204                                if(plot_errchk == 1 &&
                                        ↪ segment_counter==0){
1205                                        printf("plotting fft\n");
1206                                        double *fft_mag;//for
                                                ↪ plotting
1207                                        fft_mag = (double*)malloc(
```

```
    ↪ sizeof(double)*(
    ↪ segment_len/2+1));
1208 double avg_nsd_test=0.0;
1209 int bb;
1210 for(bb=0;bb<(segment_len
    ↪ /2+1);bb++){
1211
1212     fft_mag[bb] = (2.0*
        ↪ creal(
        ↪ segment_fft_array
        ↪ [bb])*2.0*
        ↪ creal(
        ↪ segment_fft_array
        ↪ [bb]) + 2.0*
        ↪ cimag(
        ↪ segment_fft_array
        ↪ [bb])*2.0*
        ↪ cimag(
        ↪ segment_fft_array
        ↪ [bb]))/((
        ↪ window_norm_correct
        ↪ ));//actually
        ↪ the psd
        ↪ currently,
        ↪ this is a
        ↪ convenient
        ↪ place to plot
        ↪ NSD, FFT, PSD
        ↪ for error
        ↪ checking
1213     avg_nsd_test =
        ↪ avg_nsd_test +
        ↪  fft_mag[bb];
1214 }
1215 avg_nsd_test = avg_nsd_test
    ↪ /(samp_rate/2);
1216 printf("avg_nsd_test = %e\n"
    ↪ ,sqrt(avg_nsd_test));
```

```
1217                                        plotlogplot(segment_len/2+1,
                                                ↪ fft_mag);
1218
1219                                        printf("press ENTER to
                                                ↪ continue.\n");
1220                                        getchar();
1221                                        free(fft_mag);
1222                                }
1223
1224
1225 //DETERMINE IF THERE ARE SEGMENTS LEFT TO FFT
1226 //
    ↪ ********************************************************************************
    ↪
1227
1228                                if(is_flat == 0){
1229                                        if(is_on_ch == 0){
1230                                                if(off_cntr-(cum_data
                                                        ↪ -(int)
                                                        ↪ segment_len/2)
                                                        ↪  < (int)
                                                        ↪ segment_len){
1231                                                    segs_left = 0;
                                                            ↪
                                                            ↪
                                                            ↪ }
1232                                                }
1233                                        else if(is_on_ch == 1){
1234                                                if(on_cntr-(cum_data
                                                        ↪ -(int)
                                                        ↪ segment_len/2)
                                                        ↪  < (int)
                                                        ↪ segment_len){
1235                                                    segs_left = 0;
1236                                                }
1237                                        }
1238                                }
1239                                else{//if not chopped
```

```
1240                                        if(array_len-(cum_data-(int)
                                                ↪ segment_len/2) < (int
                                                ↪ )segment_len){//Could
                                                ↪  simplify by setting
                                                ↪ off_cntr = array_len
                                                ↪ earlier in the code,
                                                ↪ but this is probably
                                                ↪ easier to debug later
1241                                                segs_left = 0;
1242                                        }
1243                                    }
1244
1245    //FIND THE POWER SPECTRUM AND ADD TO THE CUMULATIVE ARRAY AND
            ↪ NORMALIZE
1246    //
            ↪ ********************************************************************************
            ↪
1247
1248
1249
1250                                    //do the welch and normalize here
                                            ↪ and cumulative
1251                                    for(cntr=0;cntr<(int)segment_len
                                            ↪ /2+1;cntr++){
1252                                        psd_cumulative_array[cntr] =
                                                ↪  psd_cumulative_array
                                                ↪ [cntr] + 2.0*(creal(
                                                ↪ segment_fft_array[
                                                ↪ cntr])*creal(
                                                ↪ segment_fft_array[
                                                ↪ cntr]) + cimag(
                                                ↪ segment_fft_array[
                                                ↪ cntr])*cimag(
                                                ↪ segment_fft_array[
                                                ↪ cntr]))/
                                                ↪ window_norm_correct;
1253
1254                                    }
```

```
1255
1256                    //printf("psd_cumulative_array
                            ↪ [200] = %0.9f\n",
                            ↪ psd_cumulative_array[200]);
1257                    segment_counter = segment_counter
                            ↪ + 1;
1258                    //printf("segment counter is: %d\n
                            ↪ ",segment_counter);
1259                    cum_data = cum_data - (int)
                            ↪ segment_len/2; //This is the
                            ↪  0.5 overlap of Welch's
                            ↪ method.
1260                    //printf("cumulative data = %d\n",
                            ↪ cum_data);
1261
1262                }
1263                //last step of averaging all of the PSD
                        ↪ data - divide by number of segments
1264                max_psd = 0.0;
1265                min_psd = 100.0;
1266
1267                for(cntr=0;cntr<(int)segment_len/2+1;cntr
                        ↪ ++){
1268                    psd_cumulative_array[cntr] =
                            ↪ psd_cumulative_array[cntr]/
                            ↪ segment_counter;
1269                    if(psd_cumulative_array[cntr]<
                            ↪ min_psd){//find min of psd
1270                        min_psd =
                                ↪ psd_cumulative_array[
                                ↪ cntr];
1271                    }
1272                    if(psd_cumulative_array[cntr]>
                            ↪ max_psd){//find max of psd
1273                        max_psd =
                                ↪ psd_cumulative_array[
                                ↪ cntr];
1274                    }
```

```
1275
1276                            }
1277
1278                            printf("min of psd_cumulative_array is:
                                    ↪ %0.10e\n",min_psd);
1279                            printf("max of psd_cumulative_array is:
                                    ↪ %0.10e\n",max_psd);
1280
1281
1282
1283 //PLOTTING SEPARATE NOISE SOURCES
1284 //
     ↪ ********************************************************************************
     ↪
1285
1286
1287                            //plotting all noise sources separately
1288                            if(plot_separate_source == 1){
1289                                   int ggg=0;
1290                                   double **cirq_comp_noise;
1291                                   cirq_comp_noise = array2D(14, (int
                                           ↪ )(segment_len/2+1));//malloc
                                           ↪  a 2d array
1292
1293
1294                                   double *ch_gc;
1295                                   ch_gc = (double *)malloc(sizeof(
                                           ↪ double)*(segment_len/2+1));
1296
1297                                   double rrss;//rs
1298                                   if(k==0){//ch1
1299                                          printf("plotting channel 1\n
                                                  ↪ ");
1300                                          int xy;
1301                                          for(xy=0;xy<(segment_len
                                                  ↪ /2+1);xy++){
1302                                                 ch_gc[xy] =
                                                         ↪ gain_ch1_data_interp
```

```
                                    ↪ [xy];
1303                                //printf("ch_gc: %f\n
                                    ↪ ",ch_gc[xy]);
1304                            }
1305                            rrss = r_of_chans[0];
1306
1307
1308                    }
1309                    if(k==1){//ch2
1310                            printf("plotting channel 2\n
                                ↪ ");
1311                            int xy;
1312                            for(xy=0;xy<(segment_len
                                ↪ /2+1);xy++){
1313                                ch_gc[xy] =
                                    ↪ gain_ch2_data_interp
                                    ↪ [xy];
1314                            }
1315                            rrss = r_of_chans[1];
1316                    }
1317                    if(k==2){//ch3
1318                            printf("plotting channel 3\n
                                ↪ ");
1319                            int xy;
1320                            for(xy=0;xy<(segment_len
                                ↪ /2+1);xy++){
1321                                ch_gc[xy] =
                                    ↪ gain_ch3_data_interp
                                    ↪ [xy];
1322                            }
1323                            rrss = r_of_chans[2];
1324                    }
1325                    if(k==3){//ch4
1326                            printf("plotting channel 4\n
                                ↪ ");
1327                            int xy;
1328                            for(xy=0;xy<(segment_len
                                ↪ /2+1);xy++){
```

```
1329                        ch_gc[xy] =
                                ↪ gain_ch4_data_interp
                                ↪ [xy];
1330                    }
1331                    rrss = r_of_chans[3];
1332                }
1333                //double freqy;
1334                int hh;
1335                for(ggg=0;ggg<(segment_len/2+1);
                        ↪ ggg++){
1336                    g1 = sqrt(ch_gc[ggg]);
1337                    g2 = g1;
1338                    //frequency -> 0th row
1339                    cirq_comp_noise[0][ggg] =
                            ↪ ggg*(samp_rate/2.0)/(
                            ↪ segment_len/2.0+1.0);
1340                    //daq -> 1st row
1341                    cirq_comp_noise[1][ggg] =
                            ↪ daq*daq;
1342                    //r4 V -> 2nd row
1343                    cirq_comp_noise[2][ggg] =
                            ↪ 4.0*r4*KB*temp;
1344                    //r4 I -> 3rd row
1345                    cirq_comp_noise[3][ggg] = (
                            ↪ in*r4)*(in*r4);
1346                    //r3 V -> 4th row
1347                    cirq_comp_noise[4][ggg] =
                            ↪ 4.0*r3*KB*temp*(r4/r3
                            ↪ )*(r4/r3)*(g2/gainz2)
                            ↪ *(g2/gainz2);
1348                    //en 2nd stage -> 5th row
1349                    cirq_comp_noise[5][ggg] = en
                            ↪ *en*g2*g2;
1350                    //in 2nd stage -> 6th row
1351                    cirq_comp_noise[6][ggg] = (
                            ↪ in*zout)*(in*zout)*g2
                            ↪ *g2;
1352                    //r2 V -> 7th row
```

```
1353                cirq_comp_noise[7][ggg] =
                        ↪ 4.0*r2*KB*temp*g2*g2;
1354                //r2 I -> 8th row
1355                cirq_comp_noise[8][ggg] = (
                        ↪ in*r2)*(in*r2)*g2*g2;
1356                //r1 V -> 9th row
1357                cirq_comp_noise[9][ggg] =
                        ↪ 4.0*r1*KB*temp*(r2/r1
                        ↪ )*(r2/r1)*(g1/gainz1)
                        ↪ *(g1/gainz1)*g2*g2;
1358                //en 1st stage -> 10th row
1359                cirq_comp_noise[10][ggg] =
                        ↪ en*en*g1*g1*g2*g2;
1360                //in 1st stage -> 11th row
1361                cirq_comp_noise[11][ggg] = (
                        ↪ in*find_magz(
                        ↪ cirq_comp_noise[0][
                        ↪ ggg],rrss))*(in*
                        ↪ find_magz(
                        ↪ cirq_comp_noise[0][
                        ↪ ggg],rrss))*g1*g1*g2*
                        ↪ g2;
1362                //sample V (theory) -> 12th
                        ↪ row
1363                cirq_comp_noise[12][ggg] =
                        ↪ 4.0*KB*rrss*temp*g1*
                        ↪ g1*g2*g2;
1364                //total (theory) -> 13th row
1365                cirq_comp_noise[13][ggg] =
                        ↪ 0.0;
1366                for(hh=1;hh<13;hh++){//skip
                        ↪ row 0, the frequency
                        ↪ row
1367                    cirq_comp_noise[13][
                            ↪ ggg] =
                            ↪ cirq_comp_noise
                            ↪ [13][ggg] +
                            ↪ cirq_comp_noise
```

```
                    ↪ [hh][ggg];
1368            }
1369
1370
1371        }
1372
1373
1374        // + + + + + + + + + + ;
1375
1376        plotlogplotmany(14,segment_len
                ↪ /2+1,psd_cumulative_array,
                ↪ cirq_comp_noise);
1377
1378        printf("Press ENTER to continue\n"
                ↪ );
1379        getchar();
1380
1381        //Here plot the signals relative
                ↪ to input (divide out the
                ↪ gain)
1382
1383        double *
                ↪ psd_cumulative_array_rel2in;
1384        psd_cumulative_array_rel2in = (
                ↪ double *)malloc(sizeof(
                ↪ double)*(segment_len/2.0
                ↪ +1.0));
1385
1386
1387        for(ggg=0;ggg<(segment_len/2+1);
                ↪ ggg++){
1388            g1 = sqrt(ch_gc[ggg]);
1389            g2 = g1;
1390
1391            //daq -> 1st row
1392            cirq_comp_noise[1][ggg] =
                    ↪ cirq_comp_noise[1][
                    ↪ ggg]/(g1*g1*g2*g2);
```

```
1393                                    //r4 V -> 2nd row
1394                                    cirq_comp_noise[2][ggg] =
                                            ↪ cirq_comp_noise[2][
                                            ↪ ggg]/(g1*g1*g2*g2);
1395                                    //r4 I -> 3rd row
1396                                    cirq_comp_noise[3][ggg] =
                                            ↪ cirq_comp_noise[3][
                                            ↪ ggg]/(g1*g1*g2*g2);
1397                                    //r3 V -> 4th row
1398                                    cirq_comp_noise[4][ggg] =
                                            ↪ cirq_comp_noise[4][
                                            ↪ ggg]/(g1*g1*g2*g2);
1399                                    //en 2nd stage -> 5th row
1400                                    cirq_comp_noise[5][ggg] =
                                            ↪ cirq_comp_noise[5][
                                            ↪ ggg]/(g1*g1*g2*g2);
1401                                    //in 2nd stage -> 6th row
1402                                    cirq_comp_noise[6][ggg] =
                                            ↪ cirq_comp_noise[6][
                                            ↪ ggg]/(g1*g1*g2*g2);
1403                                    //r2 V -> 7th row
1404                                    cirq_comp_noise[7][ggg] =
                                            ↪ cirq_comp_noise[7][
                                            ↪ ggg]/(g1*g1*g2*g2);
1405                                    //r2 I -> 8th row
1406                                    cirq_comp_noise[8][ggg] =
                                            ↪ cirq_comp_noise[8][
                                            ↪ ggg]/(g1*g1*g2*g2);
1407                                    //r1 V -> 9th row
1408                                    cirq_comp_noise[9][ggg] =
                                            ↪ cirq_comp_noise[9][
                                            ↪ ggg]/(g1*g1*g2*g2);
1409                                    //en 1st stage -> 10th row
1410                                    cirq_comp_noise[10][ggg] =
                                            ↪ cirq_comp_noise[10][
                                            ↪ ggg]/(g1*g1*g2*g2);
1411                                    //in 1st stage -> 11th row
1412                                    cirq_comp_noise[11][ggg] =
```

```
                    ↪ cirq_comp_noise[11][
                    ↪ ggg]/(g1*g1*g2*g2);
1413                //sample V (theory) -> 12th
                    ↪ row
1414                cirq_comp_noise[12][ggg] =
                    ↪ cirq_comp_noise[12][
                    ↪ ggg]/(g1*g1*g2*g2);
1415                //total (theory) -> 13th row
1416                cirq_comp_noise[13][ggg] =
                    ↪ 0.0;
1417                for(hh=1;hh<13;hh++){//skip
                    ↪ row 0, the frequency
                    ↪ row
1418                    cirq_comp_noise[13][
                        ↪ ggg] =
                        ↪ cirq_comp_noise
                        ↪ [13][ggg] +
                        ↪ cirq_comp_noise
                        ↪ [hh][ggg];
1419                }
1420                psd_cumulative_array_rel2in[
                    ↪ ggg] =
                    ↪ psd_cumulative_array[
                    ↪ ggg]/(g1*g1*g2*g2);
1421            }
1422            printf("Plotting relative to input
                ↪ .\n");
1423            plotlogplotmany(14,segment_len
                ↪ /2+1,
                ↪ psd_cumulative_array_rel2in,
                ↪ cirq_comp_noise);
1424            printf("Press ENTER to continue\n"
                ↪ );
1425            getchar();
1426            free(psd_cumulative_array_rel2in);
1427            free(ch_gc);
1428
1429            free(cirq_comp_noise);
```

```
1430
1431                    }
1432
1433                    //Here is where the gain versus frequency
                            ↪  is taken into account. 4*Rs*kb*T*
                            ↪ gain^2 = (PSD measured) - (circuit
                            ↪ noise power), so 4*Rs*kb*T = (
                            ↪ PSD_measured - circuit_noise_power)
                            ↪ /gain^2
1434                    for(cntr=0;cntr<segment_len/2+1;cntr++){
1435                        if(k==0){//if it's ch1
1436                                psd_cumulative_array[cntr] =
                                    ↪  (
                                    ↪ psd_cumulative_array[
                                    ↪ cntr]-
                                    ↪ ch1_indep_noise_pwr[
                                    ↪ cntr])/(
                                    ↪ gain_ch1_data_interp[
                                    ↪ cntr]*
                                    ↪ gain_ch1_data_interp[
                                    ↪ cntr]);
1437                        }
1438                        else if(k==1){//if its ch2
1439                                psd_cumulative_array[cntr] =
                                    ↪  (
                                    ↪ psd_cumulative_array[
                                    ↪ cntr]-
                                    ↪ ch2_indep_noise_pwr[
                                    ↪ cntr])/(
                                    ↪ gain_ch2_data_interp[
                                    ↪ cntr]*
                                    ↪ gain_ch2_data_interp[
                                    ↪ cntr]);
1440                        }
1441                        else if(k==2){//if its ch3
1442                                psd_cumulative_array[cntr] =
                                    ↪  (
                                    ↪ psd_cumulative_array[
```

```
                                        ↪ cntr]-
                                        ↪ ch3_indep_noise_pwr[
                                        ↪ cntr])/(
                                        ↪ gain_ch3_data_interp[
                                        ↪ cntr]*
                                        ↪ gain_ch3_data_interp[
                                        ↪ cntr]);
1443                            }
1444                            else if(k==3){//if its ch3
1445                                    psd_cumulative_array[cntr] =
                                        ↪  (
                                        ↪ psd_cumulative_array[
                                        ↪ cntr]-
                                        ↪ ch4_indep_noise_pwr[
                                        ↪ cntr])/(
                                        ↪ gain_ch4_data_interp[
                                        ↪ cntr]*
                                        ↪ gain_ch4_data_interp[
                                        ↪ cntr]);
1446                            }
1447                            else{
1448                                    printf("gain correction not
                                        ↪ set up for more than
                                        ↪ 4 channels\n");
1449                                    exit(0);
1450                            }
1451                    }
1452
1453                    printf("min of psd_cumulative_array after
                        ↪  gain correction is: %0.10e\n",
                        ↪ min_psd);
1454                    printf("max of psd_cumulative_array after
                        ↪  gain correction is: %0.10e\n",
                        ↪ max_psd);
1455
1456
1457
1458                    //plotting psd averaged
```

```
1459                    if(plot_psd_cumul_array == 1){
1460                            plotlogplot(segment_len/2+1,
                                    ↪ psd_cumulative_array);
1461                            printf("Press ENTER to continue\n"
                                    ↪ );
1462                            getchar();
1463                    }
1464
1465                    //average the PSD here and get the NSD as
                        ↪  well
1466                    avg_PSD = 0;
1467                    avg_NSD = 0;
1468                    for(cntr = freq_min_ind;cntr<freq_max_ind
                        ↪ +1;cntr++){
1469                            avg_PSD = avg_PSD +
                                    ↪ psd_cumulative_array[cntr]/(
                                    ↪ freq_max_ind-freq_min_ind);
1470
1471                    }
1472
1473                    avg_NSD = sqrt(avg_PSD);
1474                    printf("writing to file\n");
1475                    if(is_on_ch == 0){
1476
1477                            fp_off = fopen(filename_nsd_off,"a
                                    ↪ ");
1478                            fprintf(fp_off,"%le\t",avg_NSD);
1479                            fclose(fp_off);
1480
1481                    }
1482                    else if(is_on_ch == 1){
1483                            fp_on = fopen(filename_nsd_on,"a")
                                    ↪ ;
1484                            fprintf(fp_on,"%le\t",avg_NSD);
1485                            fclose(fp_on);
1486                    }
1487                    printf("finished writing to file\n");
1488                    printf("avg PSD is %le\n",avg_PSD);
```

```
1489                            printf("avg NSD is %le\n",avg_NSD);
1490
1491  //SET psd_cumulative_array BACK TO ZERO
1492  //
      ↪ ********************************************************************************
      ↪
1493                            for(cntr = 0;cntr<(segment_len/2.0 + 1);
                                    ↪ cntr++){
1494                                psd_cumulative_array[cntr] = 0.0;
1495                            }
1496
1497                            //If using chopped data, you need the on
                                    ↪ part as well
1498                            if(is_flat == 0){
1499                                   is_on_ch++;
1500                            }
1501                            else{//don't need the on part here.
1502                                   is_on_ch = is_on_ch + 2;
1503                            }
1504
1505
1506                        }
1507
1508
1509
1510                        printf("it worked! congrats!\n");
1511
1512
1513                }
1514                //add a newline to the text files to represent a new
                        ↪ scan
1515
1516                fp_off = fopen(filename_nsd_off,"a");
1517                fprintf(fp_off,"\n");
1518                fclose(fp_off);
1519
1520                if(is_flat == 0 || getRconst == 1){//need to save the
                        ↪ on data if chopped, and the tev data
```

```
1521                    fp_on = fopen(filename_nsd_on,"a");
1522                    fp_tev = fopen(filename_tev,"a");
1523                    fp_Ron = fopen(filename_R_on,"a");
1524                    fp_Roff = fopen(filename_R_off,"a");
1525
1526                    fprintf(fp_on,"\n");
1527                    fprintf(fp_tev,"\n");
1528                    fprintf(fp_Ron,"\n");
1529                    fprintf(fp_Roff,"\n");
1530
1531                    fclose(fp_on);
1532                    fclose(fp_tev);
1533                    fclose(fp_Ron);
1534                    fclose(fp_Roff);
1535            }
1536
1537
1538    }
1539    //fftw_destroy_plan(plan);
1540    free(NSYNC);
1541    free(on_starts);
1542    free(off_starts);
1543    free(ch1_indep_noise_pwr);
1544    free(ch2_indep_noise_pwr);
1545    free(ch3_indep_noise_pwr);
1546    free(gain_ch1_data_interp);
1547    free(gain_ch2_data_interp);
1548    free(gain_ch3_data_interp);
1549
1550
1551    //fftw_free(ch_fft_array);
1552    if(is_flat == 0){
1553            fftw_free(ch_on_data);
1554            free(segment_tev_array);
1555            fftw_destroy_plan(plan_segment_tev);
1556            fftw_free(segment_tev_fft_array);
1557            fftw_destroy_plan(plan_resistance);
1558            fftw_free(resistance_array_time);
```

```
1559                fftw_free(resistance_array_fftd);
1560        }
1561
1562        fftw_free(ch_off_data);
1563
1564        fftw_destroy_plan(plan_segment);
1565        free(segment_data_array);
1566        fftw_free(segment_fft_array);
1567 //freeing uninterpolated gain data as it is no longer needed
1568        free(gain_ch1_data);
1569        free(gain_ch2_data);
1570        free(gain_ch3_data);
1571        free(gain_ch4_data);
1572        free(gain_file_freq_data);
1573
1574    if(tev_gain_meas == 1){
1575       free(gain_ch1_data_tev);
1576       free(gain_ch2_data_tev);
1577       free(gain_ch3_data_tev);
1578       free(gain_ch4_data_tev);
1579          free(gain_file_freq_data_tev);
1580    }
1581
1582
1583
1584        return 0;
1585
1586
1587
1588
1589
1590
1591  }
1592
1593 //Windowing function for Welch's PSD method - blackmanharris. This
      ↪ takes in the length of the welch segment, the place of the
      ↪ number in the welch segment (n), and returns a fraction to
      ↪ attenuate the number by to make a good blackmanharris window.
```

```
    ↪ So it returns a single number.
1594 void plotplot(long data_len,fftw_complex *ydata){
1595         FILE *pipefft = popen("gnuplot -persist","w");
1596         fprintf(pipefft, "set xlabel 'Index'\n");
1597         fprintf(pipefft, "set ylabel 'Value x 1e6'\n");
1598         fprintf(pipefft, "plot '-' with lines\n");
1599 
1600         int kk;
1601         for(kk=0;kk<data_len;kk++){//array_len;k++)
1602                 fprintf(pipefft, "%d %f\n",kk,creal(ydata[kk]));//
                        ↪ *1000000);
1603                 printf("kitty\n");
1604                 fprintf(pipefft, "e\n");
1605                 fprintf(pipefft, "refresh\n");
1606                 pclose(pipefft);
1607         }
1608 }
1609 
1610 void plotmultiplot(long data_len,double *ydata1, double *ydata2){
1611         FILE *pipefft = popen("gnuplot -persist","w");
1612 
1613         fprintf(pipefft, "set xlabel 'Index'\n");
1614         fprintf(pipefft, "set ylabel 'Value'\n");
1615 
1616         fprintf(pipefft, "plot '-'\n");
1617 
1618         int kk;
1619         for(kk=0;kk<data_len;kk++){//array_len;k++)
1620                 fprintf(pipefft, "%d %f\n",kk,ydata1[kk]);
1621                 fprintf(pipefft, "%d %f\n",kk,ydata2[kk]);
1622 
1623         }
1624 
1625         printf("kitty\n");
1626         fprintf(pipefft, "e\n");
1627         fprintf(pipefft, "refresh\n");
1628         pclose(pipefft);
1629 }
```

```
1630
1631 void plotlogplot(long data_len,double *ydata){
1632         FILE *pipefft = popen("gnuplot -persist","w");
1633         fprintf(pipefft, "set logscale xy 10\n");
1634         fprintf(pipefft, "set xlabel 'Frequency (Hz)'\n");
1635         fprintf(pipefft, "set ylabel 'NSD (V/sqrt(Hz))'\n");
1636
1637         fprintf(pipefft, "plot '-' with linespoints\n");
1638         int kk;
1639         for(kk=0;kk<data_len;kk++){
1640                 fprintf(pipefft, "%f %f\n",kk*(1000000.0/(data_len)),
                        ↪ ydata[kk]*1.0e15);
1641
1642         }
1643
1644         printf("kitty\n");
1645         fprintf(pipefft, "e\n");
1646         fprintf(pipefft, "refresh\n");
1647         pclose(pipefft);
1648 }
1649
1650 void plotlogplotfft(long data_len,long nyq,double *ydata){
1651         FILE *pipefft = popen("gnuplot -persist","w");
1652         fprintf(pipefft, "set logscale xy 10\n");
1653         fprintf(pipefft, "set xlabel 'Frequency (Hz)'\n");
1654         fprintf(pipefft, "set ylabel 'FFT (V)'\n");
1655
1656         fprintf(pipefft, "plot '-' with linespoints\n");
1657         int kk;
1658         printf("plotting every 1 data points\n");
1659         for(kk=0;kk<data_len;){
1660                 fprintf(pipefft, "%f %f\n",kk*((double)nyq/((double)
                        ↪ data_len)),ydata[kk]*1.0e6);
1661
1662                 kk=kk+1;
1663         }
1664
1665         printf("kitty\n");
```

```
1666        fprintf(pipefft, "e\n");
1667        fprintf(pipefft, "refresh\n");
1668        pclose(pipefft);
1669 }
1670
1671 void plotlogplot8(long data_len,double *ydata1,double *ydata2,double
        ↪ *ydata3,double *ydata4,double *ydata5,double *ydata6,double *
        ↪ ydata7,double *ydata8){
1672
1673        //I was unable to generate a plot with multiple lines of
               ↪ different color using just a c script. The data must be
               ↪  saved to a text file first
1674
1675        FILE *plotptr = fopen("data2plot.txt","w");
1676        int h;
1677        for(h=0;h<data_len;h++){
1678               fprintf(plotptr,"%f %f %f %f %f %f %f %f %f\n",h
                   ↪ *(1000000.0/(data_len)),ydata1[h]*1.0e10,ydata2[
                   ↪ h]*1.0e10,ydata3[h]*1.0e10,ydata4[h]*1.0e10,
                   ↪ ydata5[h]*1.0e10,ydata6[h]*1.0e10,ydata7[h]*1.0
                   ↪ e10,ydata8[h]*1.0e10,(1.0e10)*(ydata2[h]+ydata3[
                   ↪ h]+ydata4[h]+ydata5[h]+ydata6[h]+ydata7[h]+
                   ↪ ydata8[h]));
1679        }
1680
1681        fclose(plotptr);
1682
1683        FILE *pipefft = popen("gnuplot -persist","w");
1684        fprintf(pipefft, "set logscale xy 10\n");
1685        fprintf(pipefft, "set xlabel 'Frequency (Hz)'\n");
1686        fprintf(pipefft, "set ylabel 'PSD (V^2/Hz)'\n");
1687
1688        fprintf(pipefft,"set datafile separator whitespace\n");
1689        fprintf(pipefft, "plot 'data2plot.txt' using 1:2 title '
               ↪ Measured PSD' with lines, '' using 1:3 title 'En^2'
               ↪ with linespoints, '' using 1:4 title 'E1^2', '' using
               ↪ 1:5 title 'E2^2', '' using 1:6 title '(In|Zs|^2)', ''
               ↪ using 1:7 title 'DAQ^2', '' using 1:8 title '(InR2)^2',
```

```
            ↪  '' using 1:9 title 'Es^2', '' using 1:10 with lines lc
            ↪  rgb 'black' title 'theory all'\n");// with linespoints
            ↪ \n");
1690
1691
1692        printf("kitty\n");
1693        //fprintf(pipefft, "e\n");
1694        fprintf(pipefft, "refresh\n");
1695        pclose(pipefft);
1696        printf("closed the pipe\n");
1697 }
1698
1699
1700 void plotlogplotmany(int num_rows,long data_len,double *total_meas,
    ↪ double **cirq_noise){
1701        //I was unable to generate a plot with multiple lines of
            ↪ different color using just a c script. The data must be
            ↪  saved to a text file first
1702
1703        FILE *plotptr = fopen("data2plot.txt","w");
1704        int cl;
1705        int rw;
1706        for(cl=0;cl<data_len;cl++){
1707               fprintf(plotptr,"%f ",cirq_noise[0][cl]);//frequency (x
                   ↪ -axis)
1708               for(rw=1;rw<num_rows;rw++){
1709                      fprintf(plotptr,"%f ",cirq_noise[rw][cl]*1.0e20)
                          ↪ ;
1710               }
1711               fprintf(plotptr,"%f\n",total_meas[cl]*1.0e20);//
                   ↪ measured data
1712        }
1713
1714        fclose(plotptr);
1715
1716        FILE *pipefft = popen("gnuplot -persist","w");
1717        fprintf(pipefft, "set logscale xy 10\n");
1718        fprintf(pipefft, "set xlabel 'Frequency (Hz)'\n");
```

```
1719        fprintf(pipefft, "set ylabel 'PSD (1e20 x V^2/Hz)'\n");
1720
1721        fprintf(pipefft,"set datafile separator whitespace\n");
1722        fprintf(pipefft, "plot 'data2plot.txt' using 1:2 title 'DAQ'
               ↪ with lines, '' using 1:3 title 'R4 V' with linespoints,
               ↪  '' using 1:4 title 'R4 I', '' using 1:5 title 'R3 V',
               ↪ '' using 1:6 title 'en 2nd', '' using 1:7 title 'in 2nd
               ↪ ', '' using 1:8 title 'R2 V', '' using 1:9 title 'R2 I
               ↪ ', '' using 1:10 with lines lc rgb 'black' title 'R1 V
               ↪ ', '' using 1:11 title 'en 1st', '' using 1:12 title '
               ↪ in 1st', '' using 1:13 title 'Rs (theory)', '' using
               ↪ 1:14 title 'total (theory)', '' using 1:15 title 'total
               ↪  (meas)'\n");
1723        printf("kitty\n");
1724        fprintf(pipefft, "refresh\n");
1725        pclose(pipefft);
1726        printf("closed the pipe\n");
1727
1728 }
1729
1730
1731 //Function for calculating the complex impedence as a function of
       ↪ frequency. Inputs are the frequency of interest and the rs
       ↪ source resistance, output is the magnitude of the complex
       ↪ impedence for that frequency. Note that the C and R's besides
       ↪ Rs are fixed, so for a different circuit this funciton will
       ↪ need to be modified.
1732 double find_magz(double freq_val, double rss){
1733        double magz; //magnitude of z that will be returned
1734        double complex rg = 10e6; //resistor to ground
1735        double complex c2_z = -I/(freq_val*100e-9);//complex impedence
               ↪  of the capacitor parallel to rg
1736        rss = (double complex)rss;//convert to complex number (not
               ↪ sure if this is necessary)
1737        double complex c1_z = -I/(freq_val*1e-9);//complex impedence
               ↪ of AC coupling capacitor to DAQ
1738        double complex r3 = 1e6;//resistor to ground between c1 and
               ↪ DAQ
```

```
1739
1740        //computing the impedence
1741        double complex z1 = (1/(1/rg + 1/c2_z));//thevenin equivalent
1742        double complex z2 = z1 + rss + c1_z;//thevenin equivalent
1743        double complex z3 = 1/(1/z2 + 1/r3);//final thevanin
                ↪ equivalent
1744        magz = sqrt(creal(z3)*creal(z3) + cimag(z3)*cimag(z3));//
                ↪ getting the norm of the impedence
1745        return magz;
1746 }
1747
1748 //Interpolation function - returns interpolated y value for inputed x
        ↪  value. x_new is the x value you have and want the y value for
        ↪ , and x_base and y_base are the data you have to interpolate
        ↪ with.
1749 double linear_interpy(double x_new, double *x_base, double *y_base,
        ↪ double max_freqq){
1750        double y_new;
1751        int cntry=0;
1752        //taking care of special cases
1753        if(x_new<=x_base[0]){
1754               y_new = y_base[0];
1755        }
1756        else if(x_new > max_freqq){
1757               printf("interpolation frequency too high\n");
1758               exit(0);
1759        }
1760        else{
1761               while(x_new>x_base[cntry]){//find the location of the
                        ↪ value in x_base x_new is just larger than
1762                      cntry++;
1763
1764               }
1765               cntry = cntry-1;//back up one
1766               //y=mx+b. m=(y_base(n+1)-y_base(n))/(x_base(n+1)-x_base
                        ↪ (n)). b = y_base(n)-m*x_base(n). y_new=m*x_new+b
                        ↪ . Algebraically simplified.
1767               y_new = y_base[cntry] + ((y_base[cntry+1]-y_base[cntry
```

```
                ↪ ])/(x_base[cntry+1]-x_base[cntry]))*(x_new-
                ↪ x_base[cntry]);
1768        }
1769        return y_new;
1770    }
1771
1772    //Calculate the noise to be subtracted from the measured noise. May
        ↪ be better to make inputs global variables.
1773    double external_noise_calc(double gain_data, double daq, double r4,
        ↪ double temp, double in, double r3, double gainz2, double en,
        ↪ double zout,double r2, double r1, double gainz1, double rs ){
1774
1775        double indep_noise_pwr, g1, g2;
1776        g1 = sqrt(gain_data);
1777        g2 = g1;//Assume both stages have the same gain and rolloff (
            ↪ if rolloff used)
1778
1779        indep_noise_pwr = daq*daq + 4.0*r4*KB*temp + (in*r4)*(in*r4) +
            ↪  4.0*r3*KB*temp*(r4/r3)*(r4/r3)*(g2/gainz2)*(g2/gainz2)
            ↪  + en*en*g2*g2 + (in*zout)*(in*zout)*g2*g2 + 4.0*r2*KB*
            ↪ temp*g2*g2 + (in*r2)*(in*r2)*g2*g2 + 4.0*r1*KB*temp*(r2
            ↪ /r1)*(r2/r1)*(g1/gainz1)*(g1/gainz1)*g2*g2 + en*en*g1*
            ↪ g1*g2*g2 + (in*rs)*(in*rs)*g1*g1*g2*g2;
1780
1781        return indep_noise_pwr;
1782    }
1783
1784    //FFT window function (4-term blackmanharris)
1785    double blackmanharris(long n,long segment_length){
1786        double result;
1787        double a0,a1,a2,a3;
1788        a0 = 0.35875;
1789        a1 = 0.48829;
1790        a2 = 0.14128;
1791        a3 = 0.01168;
1792        result = a0-a1*cos(2.0*M_PI*(double)n/((double)segment_length
            ↪ -1.0))+a2*cos(4.0*M_PI*(double)n/((double)
            ↪ segment_length-1.0))-a3*cos(6.0*M_PI*(double)n/((double
```

```
            ↪ )segment_length-1.0));
1793        return result;
1794    }
1795
1796    //function to make 2D array
1797    double** array2D(int xdim, int ydim) {
1798        double** array2D_return;
1799        array2D_return = (double**) malloc(xdim*sizeof(double*));
1800        int z;
1801        for (z = 0; z < xdim; z++)
1802            array2D_return[z] = (double*) malloc(ydim*sizeof(double
                ↪ ));
1803
1804        return array2D_return;
1805    }
1806
1807    //function to find amplitude of frequency value. fft_length is #
        ↪ samples/2 +1
1808    void FindAmpAtFreq(double freq_of_interest, double fft_length, double
        ↪  nyquist_freq, double *fft_to_search, double *max_value,
        ↪ double *max_index){
1809        double max_tev = 0.0;//initialize to zero to find max voltage
            ↪ peak
1810        int max_tev_ind = 0;//find index of peak
1811        int tev_ind;
1812        for(tev_ind = (int)((freq_of_interest-5)/(nyquist_freq/
            ↪ fft_length));tev_ind<(int)((freq_of_interest+300)/(
            ↪ nyquist_freq/fft_length));tev_ind++){//search for the
            ↪ peak +/- 5 Hz around the chopper rate (in case the
            ↪ frequency was actually 43.1 Hz instead of 43 Hz, just
            ↪ to get better accuracy)
1813            if(fft_to_search[tev_ind]>max_tev){
1814                max_tev = fft_to_search[tev_ind];//find max
                    ↪ voltage in this range
1815                max_tev_ind = tev_ind;
1816            }
1817        }
1818        //Put the max values into the pointers which store these
```

```
            ↪ values
1819        *max_value = max_tev;
1820        *max_index = max_tev_ind;
1821    //printf("max_value should be: %f\n",max_tev);
1822    //printf("max_index should be: %d\n",max_tev_ind);
1823 }
```

Appendix B

PYTHON GUIS FOR INSTRUMENT CONTROL

We created GUIs in python using the tkinter package (see Figures in this Appendix), which is free and comes with most Python installations. It allows for data collection from the USB 6366 (and with minor modifications, can take data from the USB 4431). Plotting is fairly slow, and efficient data collection is not built into this GUI, so it is best for ensuring there are no peaks in the noise spectra and if the measured noise is close to the theoretical noise.

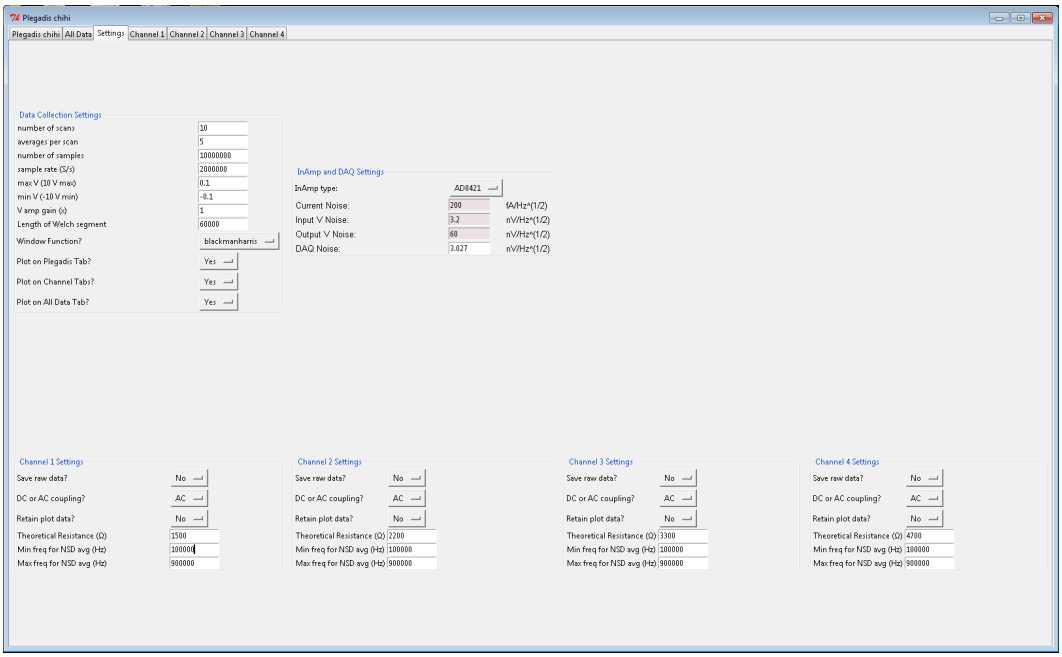

Figure B.1: Settings tab for noise measurements.

As an aside, the metal film resistors in the circuit are, from channel 1 to channel 4, 1.5 kΩ, 2.2 kΩ, 3.3 kΩ, and 4.7 kΩ, respectively. The theoretical noise, using the circuit described in the thesis, we expect to see measured noise of approximately 636, 722, 839, and 967 nV/$sqrt$Hz, respectively. The error seen between theory and measurement is then between 7-14%, which is acceptable (Kay, 2012). For our measurements, as earlier chapters describe, the noise percentage off from theory was constant. For example, by the error stated above, we mean that channel 1 was consistently too high by about 7%, **not** that the measurement was within theory by

± 7%. Thus, our measurements were very precise, but less accurate, indicating that using a constant offset to extract noise was acceptable. The RTD data and DAQ data was collected simultaneously via multiprocessing.

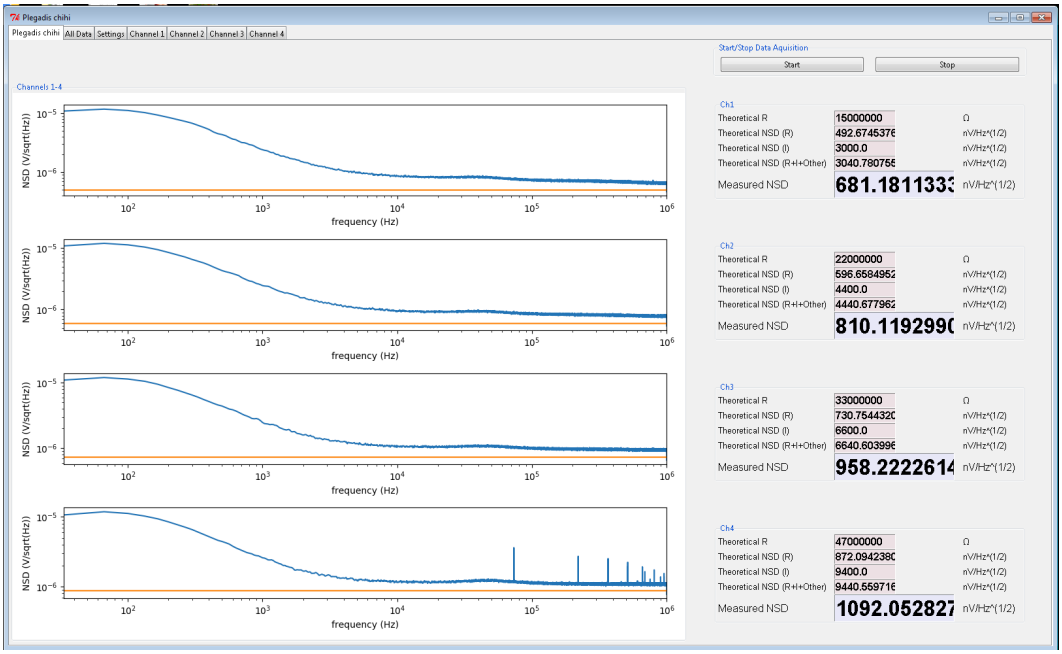

Figure B.2: Tab in GUI which shows the NSD for 4 channels. The display on the right averages the NSD bewteen values set in the settings tab. The theoretical R can be set in the settings tab, but the theoretical NSD prediction is only valid for very simple one-inamp systems, but this could be easily modified. The channel 4 spectrum is showing noise peaks from the proximity of the surface mount RTD; we had to remove the connection to this RTD for measurements due to these peaks. Physically moving the channel 4 signal traces farther from the SMT RTD would likely fix this problem.

The code for the GUI and data collection programs for the USB6366 and OM-USB-TEMP-AI is available upon request.

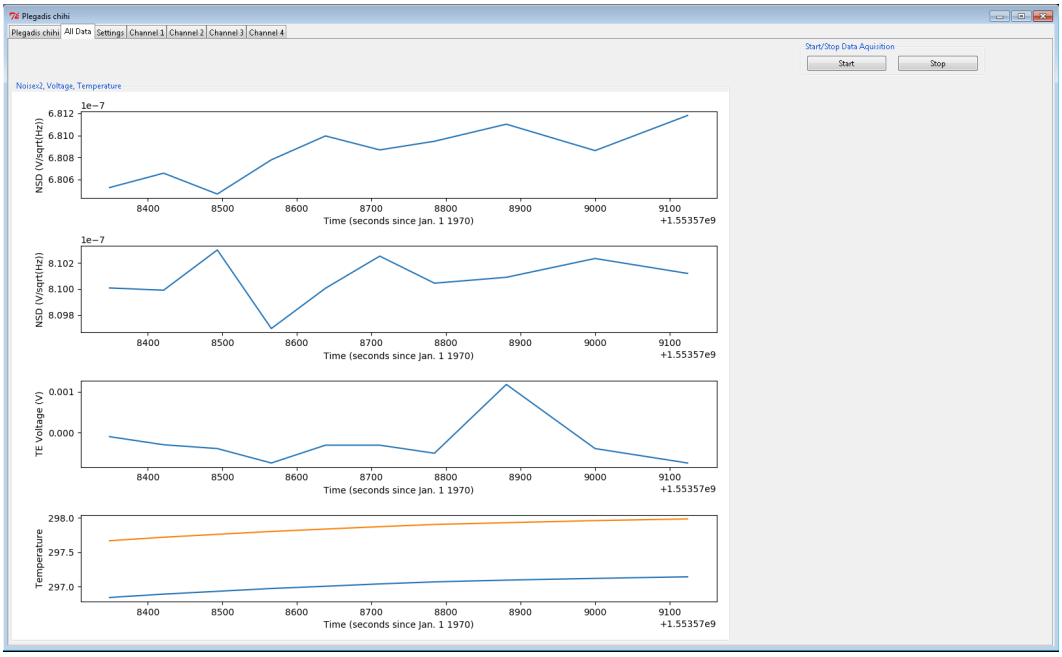

Figure B.3: Tab in GUI which shows the noise spectral density average for 2 channels, a voltage measurement between them (DC), and temperatures from the RTDs on the circuit as a function of time.

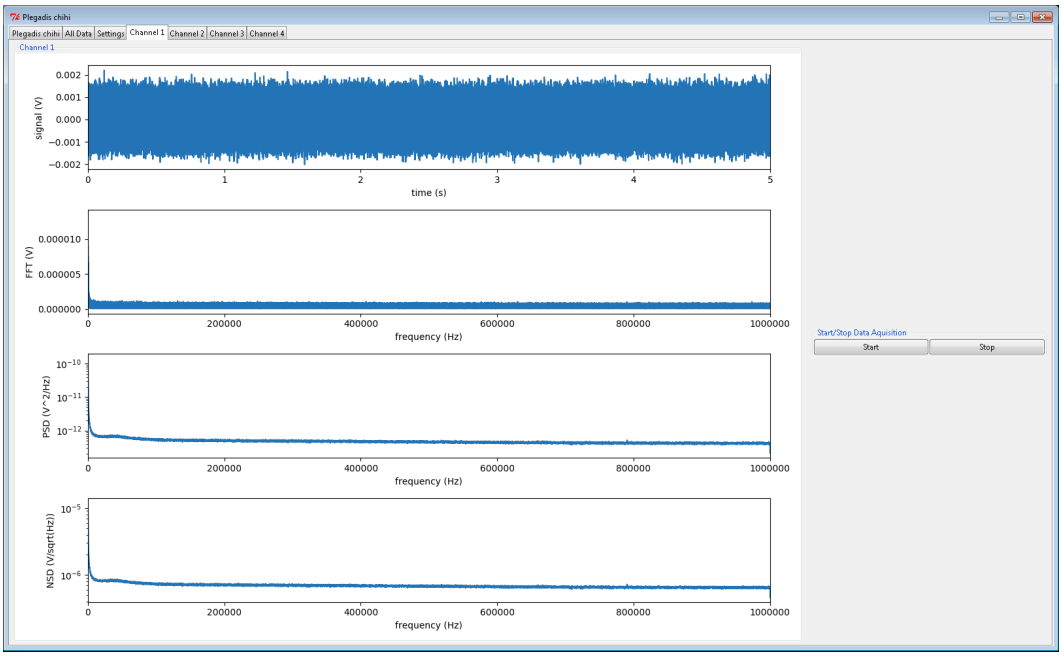

Figure B.4: Data for channel 1.

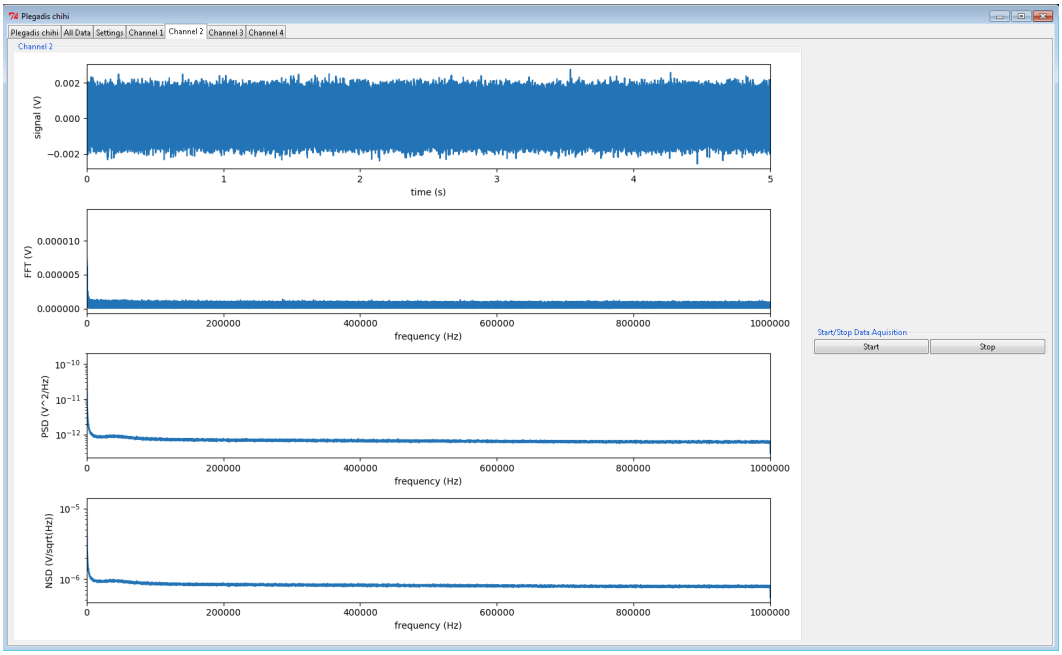

Figure B.5: Data for channel 2.

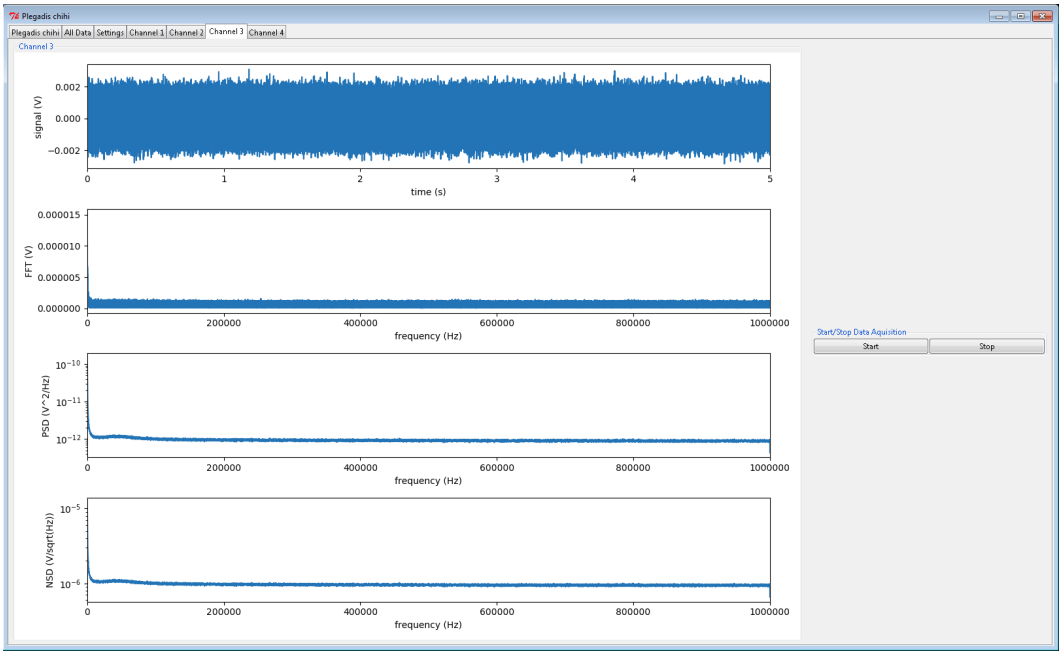

Figure B.6: Data for channel 3.

Figure B.7: Data for channel 4.

INDEX